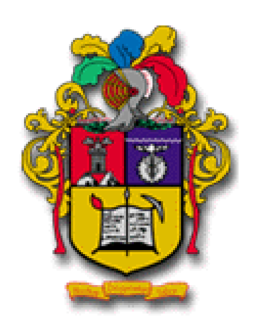


Universidad San Francisco de Quito

Colegio de Ciencias e Ingenieria

Sistemas de archivo proxy

César Izurieta

Tesis de grado presentada como requisito para la obtención del título de

Ingeniero de Sistemas

Quito, Enero del 2009

Universidad San Francisco de Quito
Colegio de Ciencias e Ingenieria

HOJA DE APROBACIÓN DE TESIS

Sistemas de archivo proxy

César Izurieta

Fausto Pasmay, MS ....
Director de Tesis (firma)

Enrique Vinicio Carrera, DSc ....
Miembro del Comité de Tesis (firma)

Fernando Romo, MS ....
Decano del Colegio Politecnico (firma)

Quito, Enero del 2009

Quisiera dedicar esta tesis a mi mamá y papá por su apoyo durante toda la vida y a mi novia Lorena por su interminable apoyo.

Agradecimientos

Quisiera agradecer a todas las personas que me ayudaron a que esta tesis se realice, en especial a Fausto Pasmay y a Vinicio Carrera que estuvieron ayudandome todo el tiempo.

Resumen

Esta tesis presenta las bases teóricas y una implementación de un sistema de archivos proxy. Un sistema de archivos proxy es un sistema de archivos que no contiene en sí archivos sino que usa otro sistema de archivos para este fin. En este proyecto, este tipo de sistemas de archivo se usará para experimentar con jerarquías alternativas de organización de archivos sin necesidad de reimplementar algoritmos de uso y acceso a disco. La implementación de este sistema de archivos se realizará usando tecnologías existentes para permitir que sea portable a varias arquitecturas y sistemas operativos. También se construirá de una manera modular para hacer posible usar diferentes configuraciones de módulos con la finalidad de que pueda ser expandido por futuros módulos. Además se ejecutarán un conjunto de pruebas para determinar cuál es el impacto de diferentes configuraciones del sistema de archivos proxy al momento de listar y acceder a múltiples archivos.

Abstract

This thesis presents the theoretical bases for the implementation of a proxy file system. A proxy file system is a file system that does not contain files itself but instead uses another file system for that purpose. In this project this kind of file system will be used to experiment with alternative hierarchies for organizing files without the need to re-implement disk-access or disk-usage algorithms. The implementation of this file system will be done using existing technologies in order to enable it to be ported easily to different architectures and operating systems. It will be also built in a modular way to make it possible to use different module configurations in order to be expandable in the future by new modules. In addition to this, several tests will be performed to determine the real performance of different configurations of the proxy file system when listing and accessing multiple files.

Índice general

Índice de figuras

Capítulo 1

Introducción

Los sistemas de archivos han estado presentes en el mundo de la informática desde hace muchas décadas. Principalmente, un sistema de archivos nos permite almacenar datos en un disco de una manera organizada. Existen muchos diferentes tipos de sistemas de archivo, pero parece que después de muchos años de desarrollo el paradigma básico de organización de archivos no ha podido evolucionar más allá de una organización jerárquica de directorios y archivos.

A partir de fines de los 90's y en especial desde el 2000 (Hen08), empezaron a aparecer ciertas aplicaciones que permiten buscar y categorizar archivos dentro de una computadora y más adelante incluso indexar varias computadoras. También los programas que manejan música, imágenes, vídeos y otros archivos multimedia empezaron a incluir dentro de sus características indexadores y sistemas de categorización. Esto fue necesario ya que las colecciones multimedia de los usuarios empezaron a crecer a un ritmo altísimo debido a varios factores como la popularización de formatos como el mp3 o sistemas de compartición de archivos tipo P2P (peer-to-peer).

Lentamente la información de organización de los archivos fue perdiendo importancia en el sistema de archivos y cada aplicación empezó a guardar su propia información al respecto tanto que muchos programas, como la mayoría de reproductores de música, prefieren tener su propia base de datos acerca de los archivos y sus características a hacer uso de una organización jerárquica en el sistema de archivos.

Por ejemplo la aplicación iTunes de Apple tiene su propia base de datos donde guarda la información de los archivos de música que maneja, y aunque permite organizar el sistema de archivos ubicando los archivos de música en una estructura definida de directorios (`Artista/Album/Pista.ext`), esta organización no es obligatoria y por tanto el sistema de archivos no tiene un papel importante en la organización y categorización de los archivos.

Existen varios intentos de generar nuevos sistemas de archivos que permitan guardar metadatos acerca de los archivos para poder luego encontrarlos más fácilmente como TagFS, Tagsistant, LFS. Estos sistemas no han llegado a popularizarse en especial por que no han podido superar a los sistemas de archivos tradicionales. Los sistemas tradicionales han pasado por un período de desarrollo muy largo, lo que les ha permitido aumentar su desempeño y fiabilidad, consolidando sus algoritmos de uso de espacio de disco y organización de la jerarquía de directorios.

1.1. Alcance

Este proyecto pretende presentar una alternativa con la cual se pueda experimentar con sistemas de archivos diferentes sin necesidad de reimplementar los algoritmos de uso de disco. Esto sería posible creando una capa de adaptación que serviría de proxy al sistema de archivos que sirve de base. A esto se lo llamará un sistema de archivos proxy, es decir un sistema de archivos que en realidad no guarda ningun archivo si no que usa un sistema de archivos existente para este fin y que permite reestructurar la organización jerárquica de los archivos de una manera dinámica. Se creará un sistema base que permita construir sobre el mismo diferentes sistemas de archivos proxy de una forma modular. Se proveerá también implementaciones de cada uno de los módulos necesarios como ejemplo. Por último se escribirá un programa con una interfaz gráfica para poder escoger diferentes módulos y montar el sistema de archivos. Todo esto se hará buscando que el sistema pueda ejecutarse en múltiples plataformas.

1.2. Tecnologías a ser usadas

A continuación se detallan algunas de las tecnologías que se usarán para poder realizar este proyecto.

1.2.1. Python

Como lenguaje base para la programación se escogió Python (pyt08). Python es un lenguaje moderno que permite crear programas rápidamente. Es un lenguaje de programación interpretado pero existen extensiones que permiten compilar el código para su ejecución óptima. Para este proyecto se usará la extensión Psyco (psy08) que compila el código dinámicamente para que se ejecute más rápido.

1.2.2. Google Code

Para mantener el código de este proyecto se escogió usar los servicios de Google Code creando un proyecto llamado “dejumble” (`http://code.google.com/p/dejumble`) donde se podrá encontrar todo el código fuente del sistema así como el código fuente para generar esta tesis, todo publicado bajo una licencia GPLv3 (gpl08). Google Code provee servicios de versionamiento basado en Subversion y almacenamiento de instaladores o paquetes.

1.2.3. FUSE

FUSE (fus08) es un conjunto de una biblioteca y un módulo de kernel para varios sistemas operativos que permite escribir sistemas de archivo a nivel de usuario y no a nivel del kernel como se lo haría tradicionalmente. Tiene tanto ventajas como desventajas. Una de las mayores desventajas es el desempeño que se ve reducido debido al paso que tienen que hacer los datos y las operaciones entre el nivel del kernel y el nivel del usuario a través de la librería de FUSE. Entre las ventajas está que el sistema de archivos corre a nivel de usuario y por tanto puede acceder a información del entorno de ejecución como por ejemplo el idioma preferido del usuario y lo que significa que también puede ser usado por cualquier usuario sin necesidad de tener privilegios.

1.2.4. Sistemas de archivo POSIX

POSIX (pos08) significa *Portable Operating System Interface*. El estándar de sistemas de archivo POSIX es el IEEE Std 1003.1. Este estándar define varias estructuras de datos relacionadas con sistemas de archivos así como también estandariza la manera en que un sistema de archivos debe reaccionar frente a ciertas acciones y comandos. Define restricciones de seguridad. La mayoría de sistemas de archivos usados hoy en día en sistemas UNIX/LINUX se apegan a este estándar.

1.2.5. XESAM

XEXAM (xes08) significa *eXtEnsible Search And Metadata specification*. Este estándar busca unificar la interfaz que usan los programas de usuario con sistemas de búsqueda de archivos. Existen varios programas como Tracker o Beagle que proveen a los usuarios con una infraestructura de búsqueda de archivos dentro de sus computadores. Cada uno de estos programas por el momento proveen interfaces gráficas y de consola para ejecutar las búsquedas, pero también implementan la interfaz XESAM para poder ser usados desde otras aplicaciones de una manera independiente.

Capítulo 2

Introducción teórica

En este capítulo se explicará en detalle como funcionan los sistemas de archivos en el sistema operativo Linux. Estos conceptos se aplican de manera similar a otros sistemas operativos.

2.1. Historia de los sistemas de archivos

En el entorno UNIX, el primer sistema de archivos apareció en 1974 y se llamaba simplemente "FS" aunque generalmente se lo denominaba S5FS, que se refiere a System V FS (Hen08). Este era un sistema de archivos basado en superbloques y inodes. Era muy lento, podía usar solamente entre el 2 % y el 5 % del ancho de banda del disco. Más adelante aparecieron sistemas de archivos como el Berkeley File System en 1974 que daba hasta cerca de un 50 % de uso del ancho de banda del disco. Este sistema fue muy utilizado y actualizado varias veces. En 1991 se creó el Log-Structured File System que daba un 70 % de eficiencia en el uso del disco al momento de escribir. Fue de los primeros en usar un Log para evitar tener que revisar todo el disco en casos de fallos. En 1993 se inventó ext2, y más adelante se actualizó a ext3. Estos sistemas usan hasta ahora un sistema de journaling y su efectividad de acceso al disco es muy alta. Más adelante aparecieron otros sistemas como XFS, JFS, ReiserFS, pero realmente un salto en utilidad lo dio ZFS que permite creación de volúmenes y discos en RAID con muy poca configuración además de poder guardar instantáneas del disco en cualquier momento, es siempre consistente por lo que no necesita un programa

para corrección de errores, puede cambiar su tamaño mientras el sistema está montado, etc. Pero aún así todos estos sistemas continúan usando los mismos conceptos de superbloque, dentries y inodes que usaba el S5FS al principio de los tiempos.

2.2. El kernel de Linux

Para esta sección se usó como guía el código fuente de Linux de la versión 2.6.27.6. (lin08)

2.2.1. Un sistema de archivos

Un sistema de archivos es el encargado de organizar archivos en un dispositivo de hardware. Generalmente los sistemas de archivos manejan archivos en discos fijos o removibles pero también existen sistemas de archivos que manejan archivos en regiones de memoria o a través de una red. En el caso que un sistema de archivos se encuentre en un medio físico, este suele tener un superbloque, es decir un sector del disco que contiene la información sobre el sistema de archivos, así el kernel sabe como montar y manejar este disco. También existen otros sistemas de archivos que permiten el acceso a información sobre el sistema operativo como *procfs* que permite tener una vista como archivos de los procesos que se están ejecutando en ese sistema.

2.2.2. El sistema de archivos virtual

Linux usa como una base para organizar los diferentes sistemas de archivos un sistema de archivos virtual (vfs08) donde se montan cada uno de los diferentes sistemas de archivos del sistema operativo. Este sistema de archivos virtual no contiene archivos ni directorios, pero mantiene información acerca de qué sistemas de archivos están montados. El sistema de archivos root (montado en `/`) es sobre el que se manejará el sistema de archivos virtual.

La estructura de los sistemas de archivos que se montan en el sistema de archivos virtual se define en el archivo `include/linux/mount.h` con el nombre de *vfsmount*. Esta estructura de datos define un *dentry* que apunta al directorio

donde se ha montado el sistema de archivos y también un puntero al superbloque del sistema de archivos. Además se define una lista de sistemas de archivos montados dentro de este sistema de archivos que se definen con la misma estructura de datos *vfsmount*.

2.2.3. Los directorios

Los directorios se definen en linux con la estructura de datos *dentry* definida en el archivo `include/linux/dcache.h`. Esta estructura de datos contiene una lista de nombres de archivos y los *inodes* relacionados, una lista de subdirectorios y los *dentries* relacionados con cada uno y un solo *dentry* padre. A todos estos se los llama hard-links ya que solo pueden apuntar a *inodes* y *dentries* dentro del mismo sistema de archivos. También existen soft-links que son solo apuntadores que pueden apuntar a cualquier lugar del sistema de archivos virtual.

2.2.4. Los archivos

La estructura de datos *inode* definida en el archivo `include/linux/fs.h` es la que se encarga de guardar información acerca de los archivos (ea99). Esta estructura no guarda información acerca del nombre del archivo ya que esta información se almacena en el *dentry*.

2.3. Limitaciones

Existen algunas limitaciones en los conceptos sobre los que se basa el sistema de archivos virtual que pueden o no limitar las capacidades de organización que pueda necesitar una aplicación:

- Se permite solamente un padre por directorio. Existe la posibilidad de crear hard-links hacia directorios desde otros directorios, pero esta práctica ha sido descalificada por presentar muchos problemas al dar la posibilidad de crear un gráfico cíclico de directorios. El único lugar donde se presenta esto es generalmente el directorio raíz para el cual su padre es sí mismo.
- Los directorios pertenecen a un sólo sistema de archivos físico.

- Se puede montar otros sistemas de archivos físicos en cualquier parte de este sistema de archivos virtual, pero un sistema de archivos solo puede controlar la jerarquía comenzando en el punto en el que está montado para adentro.
- Un archivo pertenece a un sólo sistema de archivos y puede tener varios hard-links en diferentes directorios solo dentro de su propio sistema de archivos físico. El proceso de creación y borrado de estos hard-links tiene que ser ejecutado por parte del usuario y no se mantiene automáticamente por parte del kernel o del sistema de archivos físico. Además esta característica es raramente usada por las aplicaciones comunes de escritorio.
- La mayoría de funciones que acceden al sistema de archivos reciben una ruta para manejar archivos. Aunque también muchas funciones reciben un número de archivo o "file descriptor", este es generado por el kernel y para el caso de archivos en un sistema de archivos físico este número siempre es asignado a partir de una ruta de un archivo en el sistema de archivos.
- Todas las aplicaciones de usuario que requieren leer o escribir archivos utilizan rutas en algún momento para acceder a los archivos.

Debido a todo esto es que en este proyecto se busca que el sistema de archivos proxy rompa estas limitaciones permitiendo a un archivo ubicarse en cualquier lugar del sistema de archivos proxy sin importar el sistema de archivos del cual proviene, incluso permitiendo tener diferentes archivos de diferentes sistemas de archivos en el mismo directorio.

Capítulo 3

Sistema Base

3.1. Descripción general del sistema

En Linux el sistema de archivos proxy se monta en una ruta dentro del VFS (Virtual File System). En otros sistemas operativos sucede un proceso casi idéntico. Cuando una aplicación accede a esta ruta se llama a través del kernel al módulo de FUSE que a su vez llama al sistema de archivos proxy instalado en el espacio de usuario. El sistema de archivos proxy accede al sistema de archivos original a través del VFS como se puede ver en la figura (3.1).

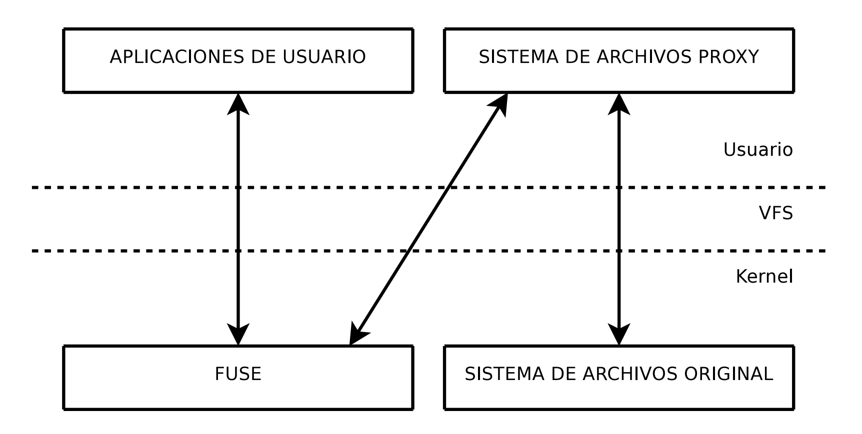


Figura 3.1: Comunicación con el kernel

Para efectos prácticos, el sistema de archivos proxy, entonces, se sitúa en medio de el sistema de archivos original y las aplicaciones de usuario como en la

siguiente figura (3.2).

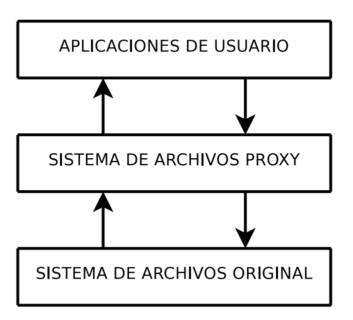


Figura 3.2: Sistema proxy

Para poder permitir una mayor modularización de un sistema de archivos proxy, se ha dividido este en varios componentes. Estos componentes son, en general, independientes unos de otros para permitir usarlos en diversas combinaciones (3.3).

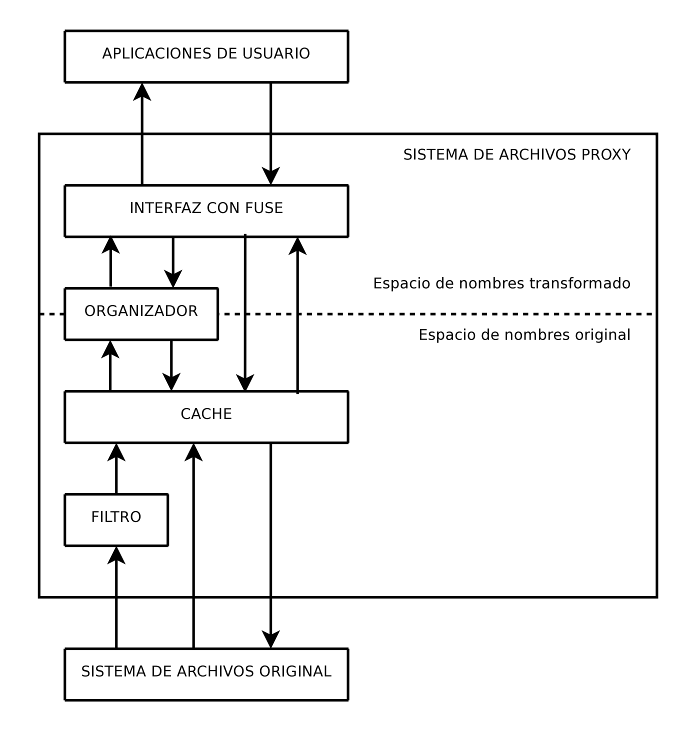


Figura 3.3: Diseño general

En este gráfico se definen dos áreas dentro del sistema de archivos proxy. La primera se define como “Espacio de nombres original”. En esta área todos los componentes tratan a los archivos con sus nombres y rutas tal y como están en el sistema de archivos original. La segunda área se denomina “Espacio de nombres transformado”. En esta área se referencia a los archivos por sus nombres y rutas una vez convertidas por el Organizador. El “Filtro” es el que escoge que archivos del sistema de archivos original se pasarán al sistema de archivos proxy. El “Cache” guarda información sobre los archivos que existen en el sistema de archivos original. El “Organizador” transforma los nombres del sistema de archivos original a rutas en el espacio de nombres transformado y viceversa. La “Interfaz con FUSE” se encarga de conectarse con el módulo del kernel de FUSE siempre con rutas en el espacio de nombres transformado. En los siguientes capítulos se explicará más a fondo acerca de cada uno de estos módulos.

De estos módulos sólo la interfaz con FUSE es la única que no puede cambiar. Para el resto de módulos se tendrá diferentes implementaciones. Un sistema de archivos proxy tiene entonces 3 módulos principales: un filtro, un cache y un organizador.

3.2. Ciclo de vida

Los siguientes puntos son los diferentes momentos por los que pasa el sistema de archivos desde que se monta hasta se desmonta.

mount Al momento de montar el sistema de archivos se ejecuta primero el archivo `dejumblefs/ui/dejumble.py` (D.1.24). Este archivo se encarga de revisar y procesar los parámetros de la línea de comando, inicializa el sistema de archivos instanciando a la clase *DejumbleFS* (D.1.10) y esta instancia la guarda en la variable global *server* para poder ser accedido por los módulos cuando lo requieran. Además se inicializa el sistema de logging.

init En el método main de la clase *DejumbleFS* (D.1.10) se inicializan los diferentes módulos que van a formar parte de este sistema de archivos. Más detalles del proceso de inicialización se encuentran en la sección (C.1).

main Una vez inicializado todo se pasa el control a la superclase *fuse.FUSE* (que es parte de la biblioteca *fuse-python*) la cual se encarga de contactarse con el módulo de FUSE en el kernel para montar el sistema de archivos.

fsinit Este es un método de la clase *DejumbleFS* (D.1.10). Inicializa el sistema de archivos. En este paso se guarda un puntero al directorio donde estamos montando para poder acceder a esos datos en caso de que se necesite. En este punto el hilo de ejecución se bloquea hasta recibir pedidos o en el momento en que se desmonta o cuando sucede un error.

request Mientras el sistema de archivos se encuentra suspendido puede recibir varios pedidos del módulo del kernel. Estos pedidos se generan cada vez que el kernel recibe pedidos a través del sistema de archivos. El pedido se recibe en la clase *DejumbleFS* (D.1.10) en cualquiera de sus funciones de sistema de archivos. Más adelante se definen cada una de estas funciones.

umount Una vez que se desmonta el sistema de archivos simplemente se termina la ejecución del programa.

fsdestroy Este es un método de la clase *DejumbleFS* (D.1.10). Se ejecuta después que el sistema de archivos ha sido desmontado. Al finalizar este método se devuelve el control al sistema operativo.

3.3. Pedidos

Para cada pedido sobre el sistema de archivos montado, la clase *DejumbleFS* (D.1.10) llama a las siguientes funciones dependiendo del pedido que se está realizando.

3.3.1. Funciones sobre la estructura del sistema de archivos

Las siguientes funciones son reenviadas al organizador definido al momento de montar el sistema de archivos.

getattr Obtiene los atributos del archivo o directorio.

readdir Obtiene una lista de archivos dado una ruta.

readlink Lee un link y devuelve la ruta real.

unlink Elimina un archivo.

rename Cambia de nombre un archivo.

3.3.2. Funciones sobre las propiedades de un archivo o directorio

Todas estas funciones se llaman directamente en el cache sin pasar por el organizador. Modifican solamente las propiedades del archivo. La única excepción es *truncate* que sí cambia al archivo. Este método requiere a nivel de FUSE para poder cortar un archivo sin abrirlo.

chmod Cambia permisos a un archivo.

truncate Corta la longitud de un archivo.

utime Actualiza la fecha de acceso al archivo.

access Prueba si se puede acceder al archivo.

3.3.3. Funciones sobre los contenidos de un archivo

Existen ciertas operaciones que se utilizan para acceder y cambiar el contenido de un archivo. Se reenvía los pedidos de estas funciones a la clase DejumbleFile existente dentro del cache que se esté utilizando.

read Devuelve datos del archivo. Recibe un parámetro que es la cantidad de bytes que se requieren.

write Escribe datos al archivo. Devuelve la cantidad de datos escritos.

release Cierra el archivo.

fsync Sincroniza los datos con el disco o medio físico.

flush Envía los datos que se encuentren en un buffer.

fgetattr Obtiene los atributos del archivo.

ftruncate Corta la longitud de un archivo. Recibe un parámetro que es la cantidad de bytes máximos que contendrá el archivo después de ejecutar esta función.

Capítulo 4

Implementaciones

En este capítulo se explicará como funcionan cada una de las implementaciones provistas. La relación entre las diferentes clases se encuentra en la figura (4.1).

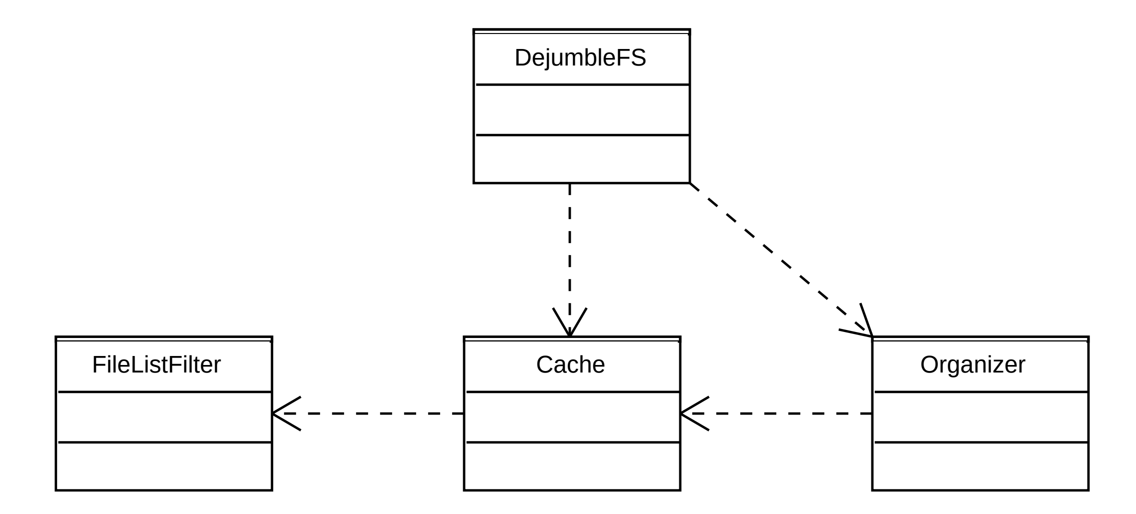


Figura 4.1: Usos de clases

4.1. Filtros

Un filtro se encarga de escoger que partes del sistema de archivos original serán pasadas al organizador. La implementación más sencilla sería usar todo el sistema de archivos original. Se puede complicar un poco más pasando solamente

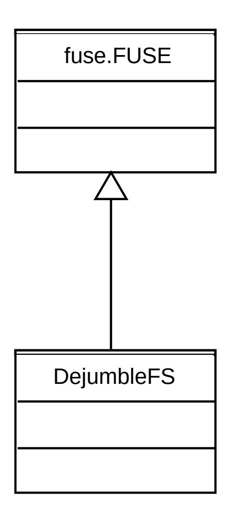


Figura 4.2: Diagrama de clases - DejumbleFS

un subdirectorio del sistema original o se puede llegar a filtros más complejos como el resultado de una búsqueda o los archivos de música que tengan en sus metadatos ciertas características.

Los filtros se encuentran en el archivo `dejumblefs/filter.py` (D.1.4). Existe una clase base llamada *FileListFilter* que tiene un sólo método público: *filelist*. Este método devuelve una lista de archivos y es únicamente llamado desde el cache. La lista que devuelve contiene ítems con la ruta completa dentro del VFS o una ruta relativa que comienza con "./" si el archivo se encuentra en el directorio original donde se montó el sistema de archivos proxy. Esto es necesario ya que una vez montado el sistema de archivos proxy cualquier llamada a su propio directorio accedería al sistema de archivos proxy y no al sistema original. Además puede hacer uso de los parámetros *query* y *root*. Estos parámetros se los define al momento de montar y pueden ser usados por cualquier implementación como información adicional para realizar el filtrado.

4.1.1. Directorio Completo

Este filtro se encuentra en el archivo `dejumblefs/filters/completedirectory.py` (D.1.5) y la clase se llama *CompleteDirectoryFileListFilter*. Simplemente pasa

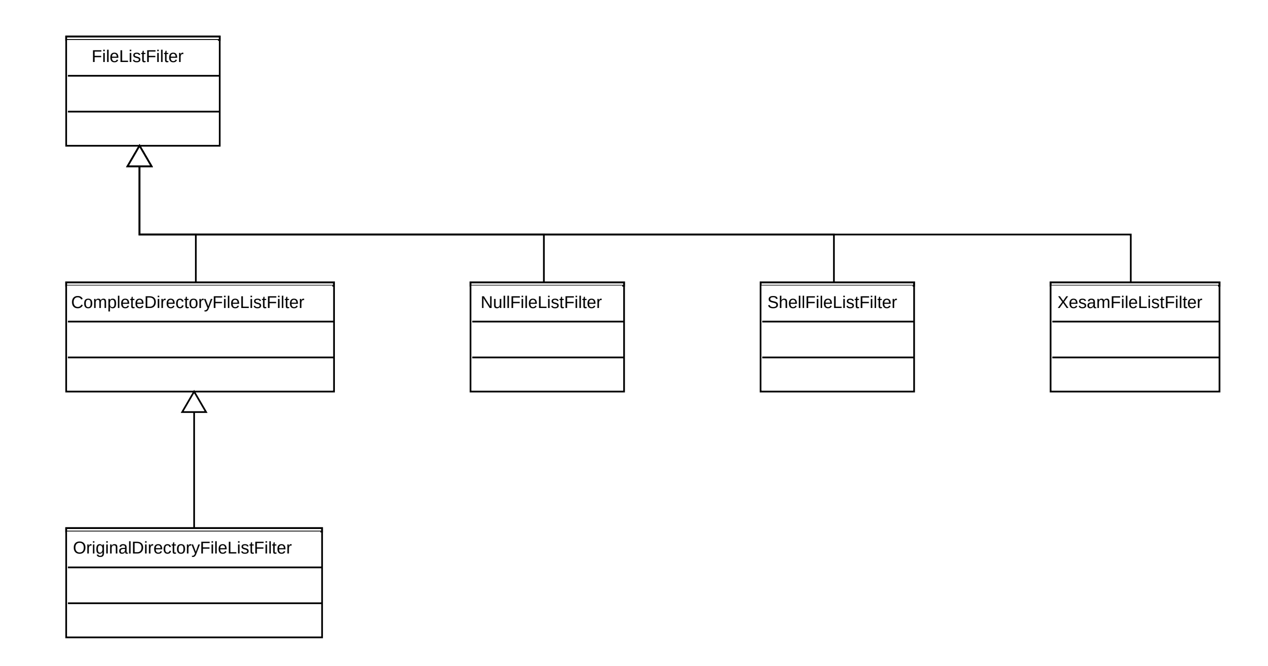


Figura 4.3: Diagrama de clases - Filtros

todos los archivos que se encuentran en el directorio determinado por la opción root. Se debe usar explícitamente con la opción nonempty al momento de montar.

4.1.2. Directorio Original

Este filtro es la implementación por defecto. Se encuentra en el archivo `dejumblefs/filters/originaldirectory.py` (D.1.7) y la clase se llama *OriginalDirectoryFileListFilter*. Es exactamente igual al anterior pero pasa la lista de archivos existentes en el directorio donde se está montando.

4.1.3. Shell

Este filtro permite ejecutar un comando de shell y se pasa el resultado de esta ejecución. Se presupone que el resultado va a dar un archivo por línea. Se encuentra implementado en el archivo `dejumblefs/filters/shell.py` (D.1.8) y la clase se llama *ShellFileListFilter*.

4.1.4. Xesam

Este filtro ejecuta un query xesam y devuelve esta lista de archivos. El código de este organizador se encuentra en el archivo `dejumblefs/filters/xesam.py` (D.1.9) y la clase se llama *XesamFileListFilter*.

4.1.5. Nulo

Este es un filtro de prueba. Solo pasa el archivo /dev/null. Se utiliza para hacer pruebas. Se encuentra en el archivo `dejumblefs/filters/null.py` (D.1.6) y la clase se llama *NullFileListFilter*.

4.2. Caches

El cache se encarga de guardar o escribir los cambios en el sistema de archivos original. El cache más simple escribe directamente los cambios al sistema de archivos original. Un cache diferente podría simplemente escribir cualquier cambio a memoria y descartarlos cuando se desmonten. Además el cache es el encargado de mantener la lista de archivos que se encuentran en el sistema de archivos proxy para que el organizador la lea.

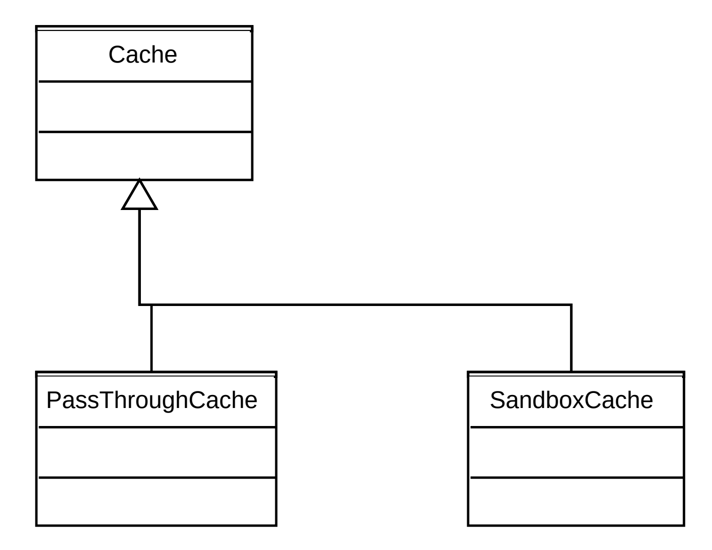


Figura 4.4: Diagrama de clases - Caches

El código para este módulo se encuentra en el archivo `dejumblefs/cache.py` (D.1.1). Consiste en una clase base llamada *Cache* que provee un conjunto de funciones para almacenar la lista de archivos que se encuentran en el sistema de archivos proxy. Además esta implementación, de la que heredan todos los caches, provee todas las funciones necesarias para acceder al sistema de archivos original y a sus archivos directamente por defecto.

4.2.1. PassThrough

Este tipo de cache simplemente reenvía todos los comandos al sistema de archivos original. Se encuentra implementado en el archivo `dejumblefs/caches/passthrough.py` (D.1.2) y la clase se llama *PassThroughCache*.

4.2.2. Sandbox

Este tipo de cache lee del disco una vez y guarda cualquier cambio solamente en memoria. Al desmontar el sistema de archivos proxy el sistema de archivos original queda sin cambios. El código para este cache se encuentra en el archivo `dejumblefs/caches/sandbox.py` (D.1.3) y la clase se llama *SandboxCache*.

4.3. Organizadores

Un organizador se encarga de tomar los archivos que fueron filtrados y los organiza en una estructura de directorios tradicional. El organizador puede ubicar un archivo en varias localizaciones de esta estructura.

El organizador debe poder hacer un mapeo de los archivos en dos direcciones, es decir debe poder conocer el nombre (o nombres) de un archivo en el espacio de nombres transformado dado el nombre del archivo en el espacio de nombres original y además poder hacerlo en la dirección contraria, saber el nombre del archivo en el espacio de nombres original dado el nombre del archivo en el espacio de nombres transformado.

En el archivo `dejumblefs/organizer.py` (D.1.11) se encuentra una clase llamada *Organizer*. Esta clase implementa los métodos básicos para crear un orga-

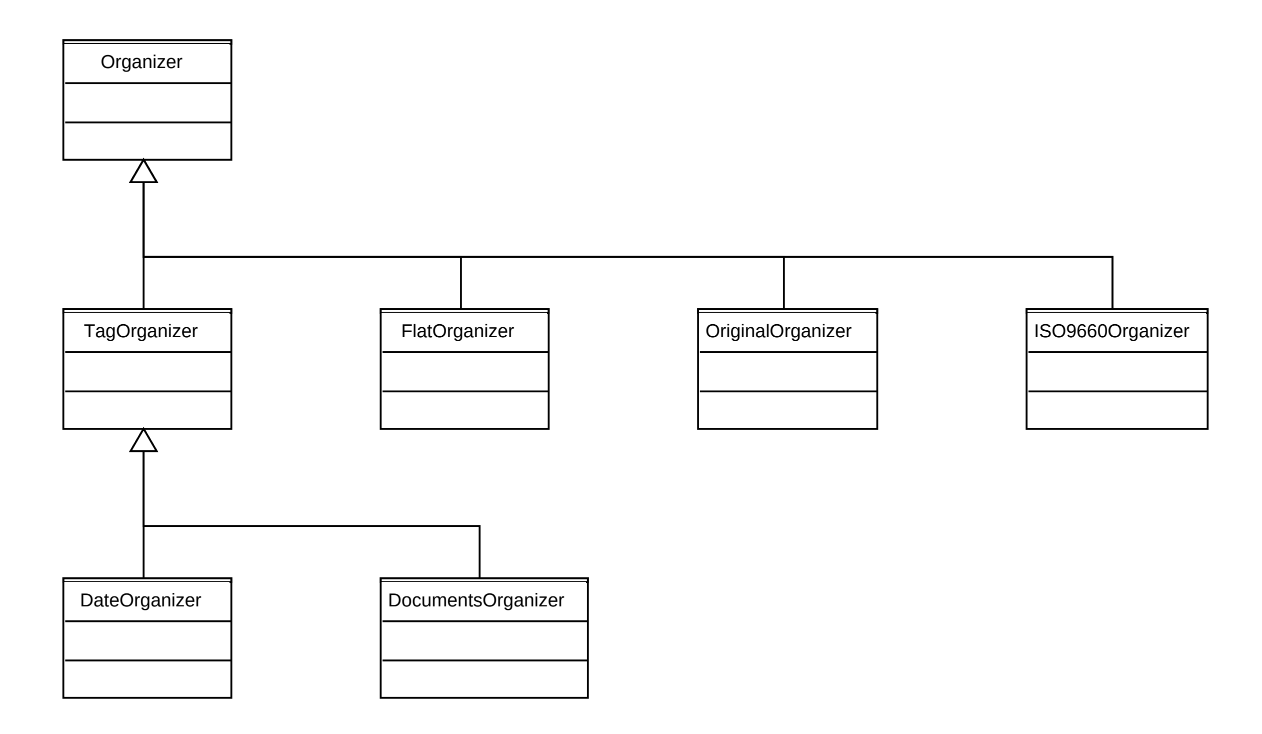


Figura 4.5: Diagrama de clases - Organizadores

nizador. Por un lado tenemos las funciones que son llamadas directamente desde el modulo de FUSE: *getattr* y *readdir*.

getattr La implementación básica de *getattr* devuelve los mismos permisos del directorio donde fue montado el sistema de archivos proxy para el caso de directorios que no existen en el sistema original o devuelve los permisos tal y como estaban en el sistema original.

readdir La función *readdir* devuelve un lista de archivo para un directorio en específico.

Además de estas funciones existen varias funciones necesarias para generar un árbol de directorios nuevo basado en el árbol de directorios original y funciones para evitar conflictos en los nombres. Las subclases de *Organizer* deben implementar dos funciones: *paths* y *dirlist*.

paths Devuelve una lista de rutas de archivos dado una ruta en el sistema de archivos original. Esta función se llama cuando el sistema se inicializa o cuando el cache expira con todos los archivos que tiene el cache.

dirlist Devuelve una lista de archivos y directorios dado un directorio en el sistema de archivos proxy. Los directorios que devuelva *dirlist* deben ser directorios que no existan en el sistema de archivos original, pero que existan en el sistema de archivos proxy.

4.3.1. Original

Presenta los archivos en la misma estructura de directorios del sistema de archivos original. La implementación se encuentra en el archivo `dejumblefs/organizers/original.py` (D.1.16) y se llama *OriginalOrganizer*. Lo que hace es atravesar todo el sistema de archivos original desde la raíz definida por el parametro *root* al momento de montar.

4.3.2. Plano

Este organizador presenta todos los archivos encontrados en un sólo directorio. La implementación se encuentra en el archivo `dejumblefs/organizers/flat.py` (D.1.14) y se llama *FlatOrganizer*. Este organizador atraviesa todo el sistema de archivos original desde la raíz definida por el parámetro *root* al momento de montar y extrae solamente el nombre de los archivos encontrados descartando la parte del directorio donde se encontraban.

4.3.3. TagOrganizer

Este organizador no se puede usar directamente, se deben usar alguna de sus subclases. La implementación se encuentra en el archivo `dejumblefs/organizer.py` (D.1.11) y se llama *TagOrganizer*. Provee la funcionalidad de poder asignar etiquetas a los archivos y luego presentar una lista de etiquetas como los directorios en la raíz del sistema de archivos montado y dentro de cada uno de estos directorios los archivos que contienen estos tags.

4.3.4. Fecha

Este organizador extiende del organizador *TagOrganizer*. Asigna tags a los archivos dada su fecha de actualización. La implementación se encuentra en el archivo `dejumblefs/organizers/date.py` (D.1.12) y se llama *DateOrganizer*. Esta clase asigna los siguientes tags:

Today Se lo asigna a todos los archivos modificados el día de hoy.

This Week Se lo asigna a todos los archivos modificados esta semana.

Last Week Se lo asigna a todos los archivos modificados la semana pasada.

YYYY MM Se lo asigna a todos los archivos dado su mes de modificación, por ejemplo “2008 March”

4.3.5. Documentos

Este organizador extiende del organizador *TagOrganizer*. Asigna tags a los archivos dada su extensión. La implementación se encuentra en el archivo `dejumblefs/organizers/documents.py` (D.1.13) y se llama *DocumentsOrganizer*. Lee un mapeo de extensiones a una descripción de tipo de archivo de un archivo de configuración. Asigna tags a los archivos dependiendo este mapeo.

4.3.6. ISO 9660

Presenta los archivos de acuerdo al estándar ISO 9660 que es muy similar al estilo que usaba DOS de 8 caracteres el nombre y 3 la extensión. La implementación se encuentra en el archivo `dejumblefs/organizers/iso9660.py` (D.1.15) y se llama *ISO9660Organizer*.

Capítulo 5

Pruebas de desempeño

En este capítulo se presentará los resultados de las pruebas de desempeño realizadas sobre el sistema de archivos proxy usando diferentes combinaciones de módulos.

5.1. Metodología

Para realizar estas pruebas se utilizó scripts de bash que ejecutan un conjunto de comandos tanto sobre el sistema de archivos original como sobre el sistema de archivos proxy montado. Estos scripts se los puede encontrar en `test_scripts/test_*`. Se utilizó una computadora MacBook Pro de 2.33 GHz Intel Core 2 Duo con 2 GB de RAM (667 MHz DDR2 SDRAM). Esta computadora corre Mac OS X 10.5.4 con todos los últimos parches disponibles al 1 de Septiembre del 2008. Además corre MacFUSE versión 1.7 y Python versión 2.5.1. Para realizar los gráficos se usó gnuplot versión 4.2 patchlevel 3. Todos los fuentes para la generación de los gráficos se encuentran en `docs/thesis/Chapter4/Chapter4Figs/*.gnu` (D.2.1). Los archivos de datos con los que se generó estos gráficos también están en el mismo directorio con la extensión `.dat`.

Cada prueba se ejecutó 10 ocasiones. Se calculó un porcentaje promedio de penalidad en el tiempo de respuesta de usar el sistema. Se escogió una de las 10 ejecuciones y se uso como fuente para los gráficos. El método de cálculo del promedio fue el siguiente:

Dados los resultados tanto para el sistema de archivos original (llamado a en adelante), y los resultados del sistema proxy (llamado b en adelante), se calculó un valor c como la ecuación (5.1) que representa el porcentaje de cambio del tiempo de ejecución de la prueba.

$$c_i = \frac{b_i}{a_i} \tag{5.1}$$

El porcentaje de cambio sería entonces lo expuesto en la ecuación (5.2).

$$porcentaje_i = 100(c_i - 1) \tag{5.2}$$

Además se hizo una regresión lineal usando la ecuación (5.3) entre c y x, donde x es el valor en el cual varió la prueba. Para la prueba A es el número de archivos y para la prueba B es el tamaño del archivo.

$$\hat{b} = \frac{\sum_{i=1}^{N}(x_i - \bar{x})(c_i - \bar{c})}{\sum_{i=1}^{N}(x_i - \bar{x})^2} \tag{5.3}$$

Debido a que este valor indica la pendiente de la curva del porcentaje de cambio del tiempo de ejecución, nos indica si existe un aumento que depende de x. Para efectos de simplificación del problema se descartó este valor si era menor al 0.1 %. En tal caso suponemos que el uso del sistema de archivos proxy tiene una penalidad en el tiempo de ejecución en el orden de O(1) con respecto al uso del sistema original. De no ser así asumimos que el sistema de archivos proxy tiene una penalidad en el orden de O(n) con respecto al sistema de archivos original.

5.1.1. Prueba A

La prueba A consiste en crear y borrar un número de archivos vacíos. En esta prueba se varió el número de archivos entre 50 y 500 con incrementos de 50 archivos. Con esta prueba se intenta analizar el impacto que tiene el sistema de archivos proxy con el manejo de estructuras de directorios en general.

5.1.2. Prueba B

La prueba B consiste en crear y borrar un archivo de un tamaño determinado. En esta prueba se varió el tamaño del archivo creado entre 20 y 200 MB con incrementos de 20 MB. Esta prueba tiene como finalidad analizar el impacto del sistema de archivos proxy en tiempo de acceso a disco con relación al sistema de archivos original.

5.2. Resultados configuración 1

Para el primer set de resultados se usó lo siguientes ajustes al momento de montar el sistema de archivos proxy.

filter = CompleteDirectory - Para acceder a un directorio completo

root = /path - Esta ruta apunta hacia un directorio vacío en el sistema de archivos original

cache = PassThrough - Esta es la opción por defecto

organizer = Original - Esta es la opción por defecto

5.2.1. Prueba A-1

Estos resultados corresponden a la ejecución de la prueba A usando la configuración 1. La media de penalización de usar esta configuración fue de aproximadamente 91 % con un factor c de 1.906 y el resultado está graficado en la figura (5.1).

5.2.2. Prueba B-1

Estos resultados corresponden a la ejecución de la prueba B usando la configuración 1. La media de penalización de usar esta configuración fue de aproximadamente 0 % con un factor c de 1.004 y el resultado está graficado en la figura (5.2).

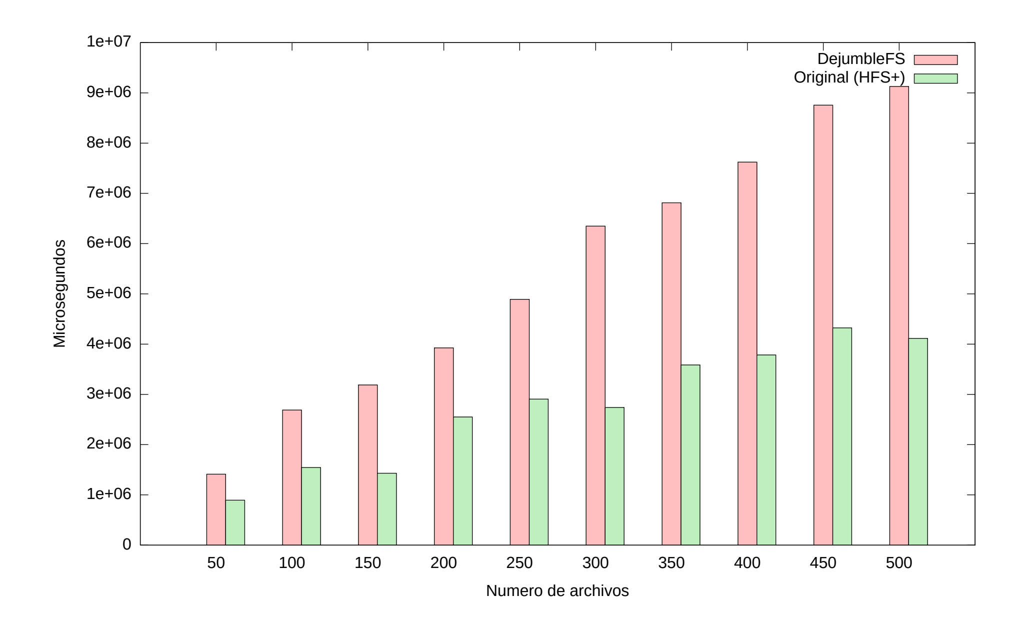


Figura 5.1: Resultados de la prueba A configuración 1

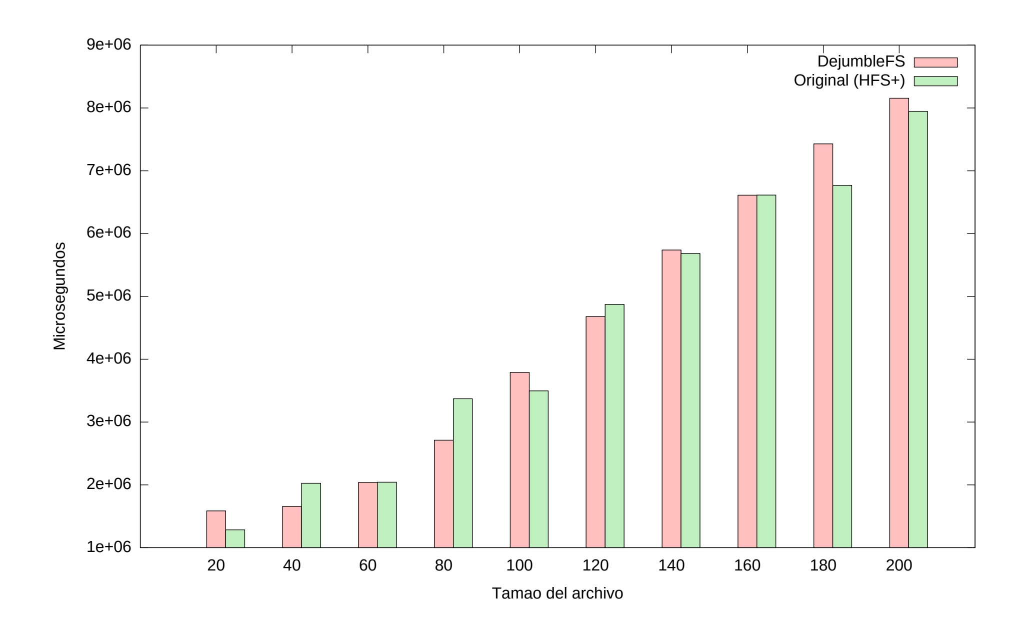


Figura 5.2: Resultados de la prueba B configuración 1

5.3. Resultados configuración 2

Para el primer grupo de resultados se usó los siguientes ajustes al momento de montar el sistema de archivos proxy.

filter = CompleteDirectory - Para acceder a un directorio completo

root = /path - Esta ruta apunta hacia un directorio vacío en el sistema de archivos original

cache = PassThrough - Esta es la opción por defecto

organizer = Date - Esta opción extiende de TagOrganizer igual que Documents

5.3.1. Prueba A-2

Estos resultados corresponden a la ejecución de la prueba A usando la configuración 2. La media de penalización de usar esta configuración fue de aproximadamente 144 % con un factor c de 2.438 y el resultado está graficado en la figura (5.3).

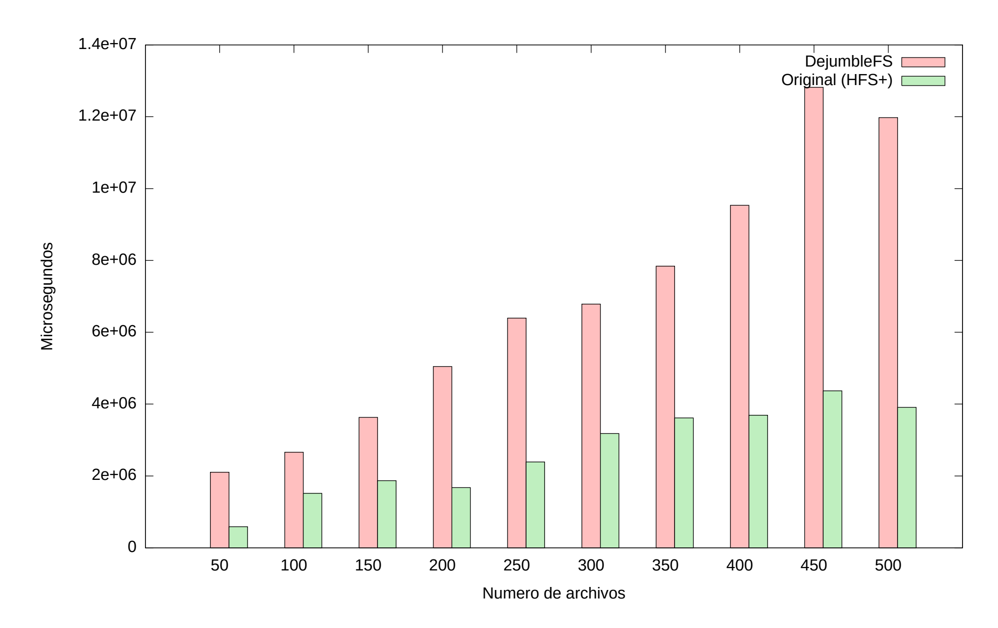


Figura 5.3: Resultados de la prueba A configuración 2

5.3.2. Prueba B-2

Estos resultados corresponden a la ejecución de la prueba B usando la configuración 2. La media de penalización de usar esta configuración fue de aproximadamente 1 % con un factor c de 1.001 y el resultado está graficado en la figura (5.4).

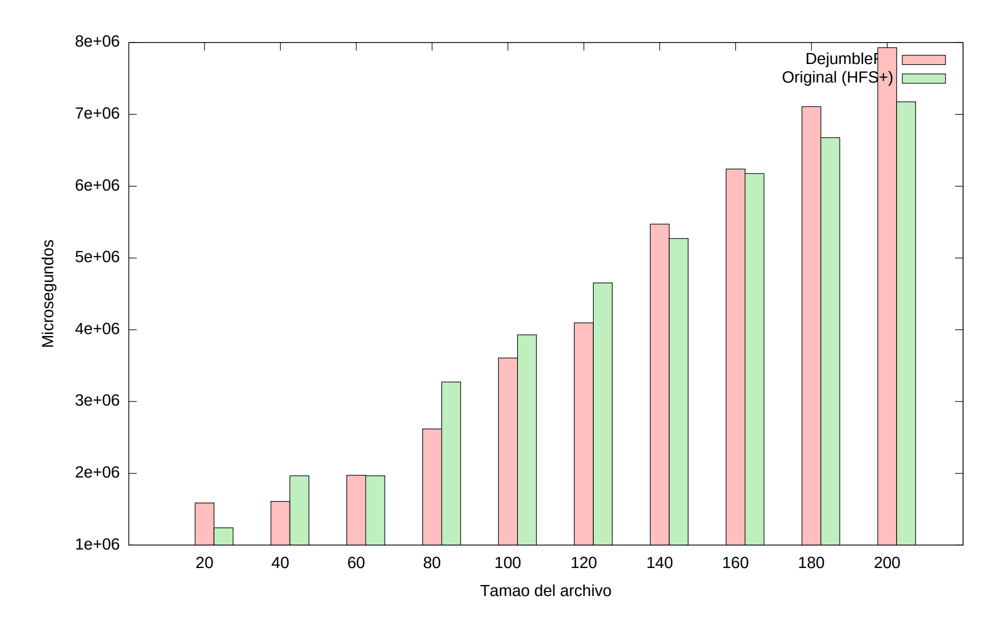


Figura 5.4: Resultados de la prueba B configuración 2

5.4. Resultados configuración 3

Para el primer set de resultados se usó lo siguientes ajustes al momento de montar el sistema de archivos proxy.

filter = CompleteDirectory - Para acceder a un directorio completo

root = /path - Esta ruta apunta hacia un directorio vacío en el sistema de archivos original

cache = PassThrough - Esta es la opción por defecto

organizer = ISO9660 - Esta opción no extiende de TagOrganizer

5.4.1. Prueba A-3

Estos resultados corresponden a la ejecución de la prueba A usando la configuración 3. La media de penalización de usar esta configuración fue de aproximadamente 98 % con un factor c de 1.976 y el resultado está graficado en la figura (5.5).

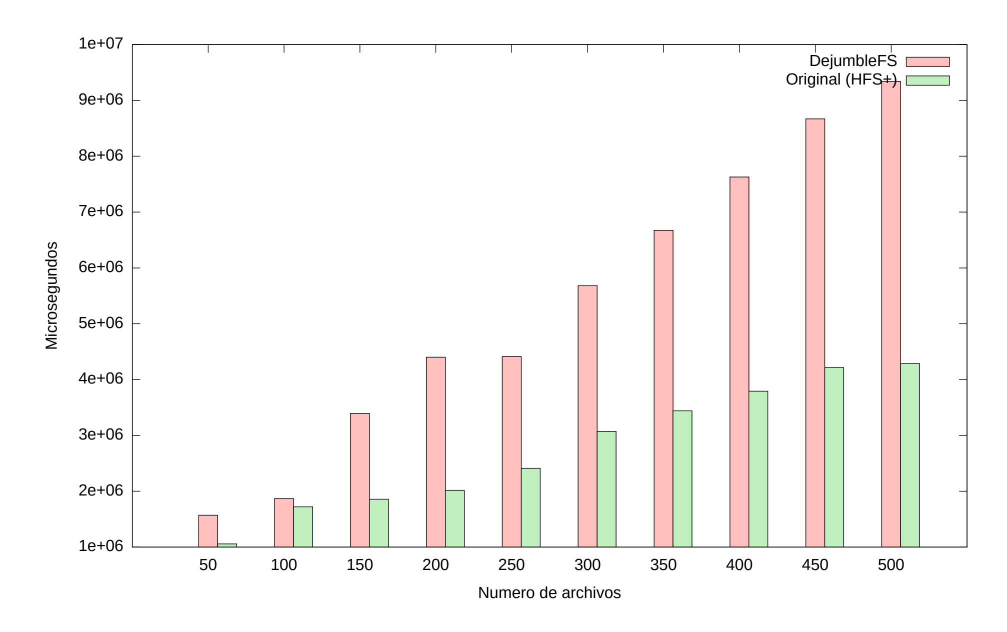


Figura 5.5: Resultados de la prueba A configuración 3

5.4.2. Prueba B-3

Estos resultados corresponden a la ejecución de la prueba B usando la configuración 3. La media de penalización de usar esta configuración fue de aproximadamente 1 % con un factor c de 1.010 y el resultado está graficado en la figura (5.6).

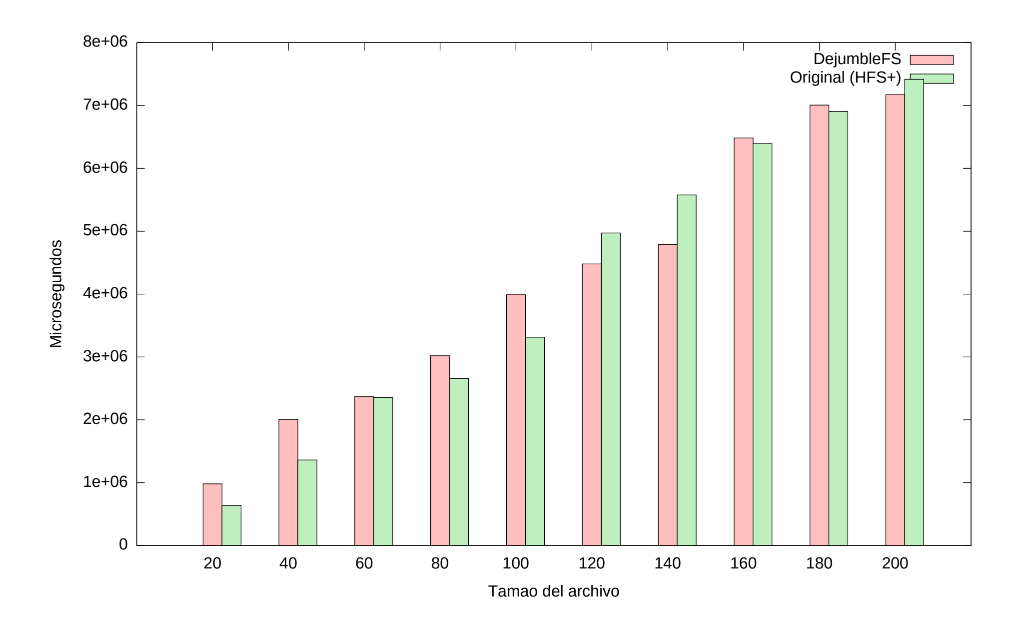


Figura 5.6: Resultados de la prueba B configuración 3

Capítulo 6

Conclusiones

6.1. Conclusiones generales

Luego de implementar y poner a prueba este sistema se puede concluir que es factible la creación de sistemas de archivos proxy. El proceso de creación de nuevos módulos del sistema es sencillo ya que pueden ser escritos en muy pocas líneas. Por ejemplo el promedio de lineas de los filtros implementados es de 6.2 líneas de código y de 15 líneas para los organizadores. Los resultados de las pruebas son mixtos con resultados excelentes en las pruebas de creación de archivos con contenido y resultados buenos en la creación y borrado de múltiples archivos. Además se demostró que se pudo crear implementaciones completas de la funcionalidad de un sistema de archivos proxy.

6.2. Resultados de creación y borrado de archivos

En el caso de manejo de listas de archivos crecientes, el tiempo de respuesta se altera linealmente con respecto al tiempo de respuesta del sistema de archivos original, es decir crece en orden O(n). El tiempo que le toma al sistema de archivos proxy crear y borrar archivos es en promedio el doble que si se usa el sistema de archivos normal.

6.3. Resultados de creación de archivos con contenido

En el caso de crear archivos con contenido la penalización que se sufre por usar el sistema de archivos proxy en todos los casos fue prácticamente nula. Esto se debe a que todas las operaciones se las pasa directamente al sistema de archivos original sin alteraciones.

6.4. Recomendaciones

En un uso cotidiano de diferentes configuraciones del sistema de archivos proxy no es muy notable el tiempo que toma crear o borrar un archivo. Estas operaciones son en general muy pocas para usuarios normales. En cambio la mayoría de usuarios acceden a sus archivos muchas veces y cambian su contenido constantemente.

Existen ciertas áreas del sistema de archivos que se modifican constantemente, creando y borrando archivos como por ejemplo los archivos temporales de internet. En este caso no sería práctico usar un sistema de archivos proxy en esa localización por dos razones. Primero, es un área que los usuarios normalmente no acceden directamente si no accede solamente el navegador y, segundo, el navegador espera una estructura especial para esa área y al usar un sistema de archivos proxy seguramente estaríamos cambiando esa estructura de directorios y archivos y el navegador no encontraría los archivos temporales.

Sin embargo existen otras áreas del sistema de archivos que los usuarios acceden constantemente como son las carpetas donde se guardan documentos o imágenes. Estas carpetas pueden hacer uso de un sistema de archivos proxy para organizarse mejor. Habría que tomar una precaución que sería en el caso de que se desee copiar muchos archivos de un disco externo u otra localización a estas carpetas. En ese caso sería mejor desmontar temporalmente el sistema de archivos proxy para hacer la copia y luego volverlo a montar ya que en este caso si estaríamos creando muchos archivos y el sistema de archivos proxy puede afectar en el desempeño.

En todo caso sería interesante poder hacer un estudio de usabilidad de los sistemas de archivo proxy con un grupo de usuarios no técnicos en el que se mida también patrones de uso del sistema de archivos en una sesión de usuario normal.

Apéndice A

Compilación del código y ejecución

Aquí están algunas guías para compilar y ejecutar el código fuente para este proyecto.

A.1. Paquetes necesitados

A.1.1. Python

Para instalar Python ver la página web `http://www.python.org`. Se recomienda la última versión. MacOS 10.5 viene con un paquete de Python adecuado y en Linux generalmente se puede usar el manejador de paquetes de la distribución para instalarlo si no está instalado por defecto.

A.1.2. FUSE

No se necesita instalar FUSE en Linux ya que es parte del kernel desde la versión 2.6.14. En MacOS X se recomienda usar MacFUSE (`http://code.google.com/p/macfuse/`). Ver el sitio web para más información. Al momento no existe una implementación completa para plataformas Windows.

A.1.3. Módulos de Python

Los siguientes módulos de Python son necesarios para ejecutar la aplicación:

1. setuptools

2. fuse-python
3. psyco
4. PyDbLite (http://quentel.pierre.free.fr/PyDbLite/index.html)

Para instalarlos se puede usar easy_install (se instala automáticamente con Python) de la siguiente manera:

```
easy_install [nombre_del_paquete]
```

Para PyDbLite existe una carpeta llamada `support/PyDbLite` donde se puede usar los mismos comandos que se presentan a continuación para instalarlo.

A.2. Compilación e instalación

Para compilar simplemente hay que ejecutar:

```
python setup.py build
```

Para instalar:

```
sudo python setup.py install
```

A.3. Pruebas

Existe un script en el directorio `test_scripts` que ejecuta estos comandos y además ejecuta los pruebas de unidad que existen. Para usar este script simplemente llamar:

```
test_scripts/build
```

A.3.1. Proctor

Se recomienda Proctor (http://www.doughellmann.com/projects/Proctor/) para ejecutar las pruebas de la aplicación.

Apéndice B

Manual de usuario

B.1. Uso desde la consola

Para usar el programa desde la consola se debe ejecutar:

```
dejumble [punto_de_montaje] [-o opción1=valor1,opción2=valor2,...]
```

El *punto_de_montaje* es el directorio donde se va a montar el sistema de archivos. Se pueden usar las siguientes opciones:

filter Define que filtro va a ser usado. Este valor debe ser un paquete de Python válido que exista dentro del paquete dejumblefs.filters. Dentro de este paquete debe existir una clase que tenga ese nombre agregando FileListFilter. Su valor por defecto es *OriginalDirectory*.

root Define el directorio base, el directorio original, sobre el cual se va a ejecutar el filtrado. Su valor por defecto es el directorio donde se va a montar. En caso de que no se modifique *root* en linux se debe usar el parámetro *nonempty* para que el sistema de archivos pueda acceder al sistema de archivos original.

query Define el query que va a ejecutar el filtro.

cache Define que cache va a ser usado. Este valor debe ser un paquete de Python válido que exista dentro del paquete dejumblefs.caches. Dentro de este paquete debe existir una clase que tenga ese nombre agregando Cache. Su valor por defecto es *PassThrough*.

organizer Define que organizador va a ser usado. Este valor debe ser un paquete de Python válido que exista dentro del paquete dejumblefs.organizers. Dentro de este paquete debe existir una clase que tenga ese nombre agregando Organizer. Su valor por defecto es *Original*.

nonempty Permite en linux que el sistema de archivos pueda acceder al sistema de archivos original.

noappledouble Evita que el sistema de archivos proxy reciba llamadas del sistema de archivos OS X de Apple acerca de archivos especiales encontrados en volúmenes de archivos normales. Esto incrementa ligeramente el desempeño en este sistema operativo.

Existen otras opciones que son propias de FUSE. Una lista se puede obtener en la página http://code.google.com/p/macfuse/wiki/OPTIONS. Esta información también se la puede obtener ejecutando:

```
dejumble --help
```

B.1.1. Ejemplos

```
dejumble punto_de_montaje -o nonempty

dejumble punto_de_montaje -o filter="Null",nonempty

dejumble punto_de_montaje -o filter="OriginalDirectory" \
                          -o organizer="Documents" \
                          -o nonempty

dejumble punto_de_montaje -o root="~/Music" \
                          -o query="find ~/Music/ -name *.mp3" \
                          -o filter="Shell" \
                          -o organizer="Flat" \
                          -o nonempty
```

B.2. Uso de la interfaz visual

Para usar la interfaz visual del programa se debe ejecutar:

`dejumblegui`

Se puede también escoger la aplicación *DejumbleFS Mounter* del manejador de aplicaciones del sistema operativo. La interfaz visual permite montar un sistema de archivos tal como lo hace la interfaz de línea de comando con las opciones específicas indicadas anteriormente pero no permite usar opciones arbitrarias de FUSE.

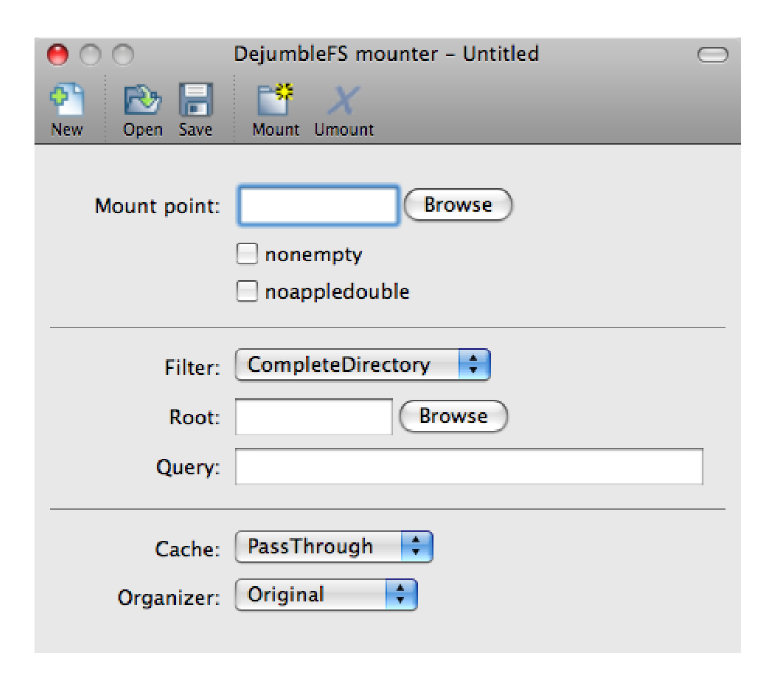


Figura B.1: Interfaz visual en el sistema operativo OS X

Esta interfaz es multi-plataforma. Se la puede usar en cualquier sistema operativo que tenga Python.

Apéndice C

Manual Técnico

C.1. Detalles del ciclo de vida de inicialización de los módulos

Los módulos se inicializan en este orden: primero el filtro, luego el cache y finalmente el organizador.

El filtro recibe en su constructor dos parámetros: *query* y *root*. Estos son asignados a variables de instancia.

El cache se inicializa con la instancia del filtro como parámetro y lo asigna a una variable de instancia.

El organizador se inicializa con dos parámetros: la instancia del cache y *recursive*. Estos se asignan a variables de instancia. *recursive* es un parámetro interno que se lo pasan las implementaciones en específico que extienden de esta clase. Más adelante se presenta este tema a mayor profundidad.

Una vez inicializados se llama al método *reset* del organizador y este resetea primero el cache.

En el cache cuando se llama el método *reset* se crea una base de datos en memoria de los archivos en el espacio de nombres original y se rellena con todos las rutas devueltas por el filtro.

En el organizador en el método *reset* crea una base de datos en memoria del mapeo entre los archivos en el espacio de nombres original y los archivos en el espacio de nombres transformado y finalmente llena su base de datos llamando el método *generateallpaths*. Este método pasa cada una de las rutas en el espacio

de nombres original que tiene el cache al método *generatepaths* que devuelve las rutas en el espacio de nombres transformado. Si se construyó el organizador con *recursive=True* también pasa cada uno de los subdirectorios que se encuentran en cada ruta. Por ejemplo para "`/tmp/directorio/subdir1/subdir2/subdir3/unarchivo.txt`" y con *root*=`/tmp/directorio/` se pasaría esta lista de archivos:

- `/tmp/directorio/subdir1/`
- `/tmp/directorio/subdir1/subdir2/`
- `/tmp/directorio/subdir1/subdir2/subdir3/`
- `/tmp/directorio/subdir1/subdir2/subdir3/unarchivo.txt`

Si *recursive=False* solamente el último ítem de esta lista. Estas rutas serían transformados dependiendo el organizador en uso. Si tomamos como ejemplo el organizador *Flat* (D.1.14) lo que se encontraría en la base de datos en memoria del organizador al final para este archivo sería lo que se ve en el cuadro (C.1).

Cuadro C.1: Base de datos en memoria para organizador Flat

realpath	path	dirname
`/tmp/.../unarchivo.txt`	`/unarchivo.txt`	`/`

En caso de usar el organizador *Original* (D.1.16) en la tabla (C.2) se puede ver el resultado.

Cuadro C.2: Base de datos en memoria para organizador Original

realpath	path	dirname
`/tmp/.../unarchivo.txt`	`/subdir1/.../unarchivo.txt`	`/subdir1/.../subdir3/`

Como último ejemplo en caso de usar el organizador *ISO9660* (D.1.15), el resultado sería lo que está en la tabla (C.3).

Cuadro C.3: Base de datos en memoria para organizador ISO9660

realpath	path	dirname
`/tmp/.../unarchivo.txt`	`/SUBDIR1/.../UNARCH~1.TXT`	`/SUBDIR1/.../SUBDIR3/`

C.2. Extendiendo DejumbleFS

C.2.1. Creando un filtro

Para crear un filtro se debe crear un archivo con el nombre del filtro en minúsculas todo, en la carpeta `dejumblefs/filters/` con extensión `.py`. En este archivo se debe crear una clase Python que tenga por nombre el nombre del filtro terminando en *FileListFilter* que extienda de *dejumblefs.filter.FileListFilter*.

Esta clase debe contener al menos un método llamado *filelist*. Este método debe devolver un iterador, generador, lista o tupla conteniendo rutas de archivos. El filtro puede hacer uso de las variables *query* y *root* para refinar su funcionamiento. Como ejemplo se puede ver `dejumblefs/filters/null.py` (D.1.6).

C.2.2. Creando un cache

Para crear un cache se debe crear una archivo con el nombre del cache en minúsculas todo, en la carpeta `dejumblefs/caches/` con extensión `.py`. En este archivo se debe crear una clase Python que tenga por nombre el nombre del cache terminando en *Cache* que extienda de *dejumblefs.cache.Cache*.

Esta clase puede sobreescribir cualquier método de la clase Cache pero en especial sobreescribiendo *getfdandfile* se puede crear un cache con una funcionalidad diferente.

Este método recibe como parámetros el path en el sistema de archivos transformado, varias banderas y el modo en el que se requiere abrir un archivo. Entre las banderas posibles se encuentran las banderas que usa la función open de C estándar como O_CREAT, O_RDONLY, etc.

El modo es una cadena que contiene "r" si se requiere escribir en el archivo, "w" si se requiere escribir "w+" si se requiere leer y escribir. Si se requiere que no

se sobreescriba el archivo si existe y se quiere agregar contenido se puede recibir “a” en vez de “w”.

El método debe devolver un descriptor o número de archivo POSIX y un *objeto tipo archivo* de Python.

Es posible sobreescribir la clase interna *Cache.DejumbleFile* que se encarga de acceder a los archivos para dar más flexibilidad al cache que se quiera implementar.

Como ejemplo se puede ver `dejumblefs/caches/sandbox.py` (D.1.3).

C.2.3. Creando un organizador basado en etiquetas

Para crear un organizador basado en etiquetas se debe crear una archivo con el nombre del organizador en minúsculas todo, en la carpeta `dejumblefs/organizers/` con extensión `.py`. En este archivo se debe crear una clase Python que tenga por nombre el nombre del organizador terminando en *Organizer* que extienda de *dejumblefs. organizer.TagOrganizer*.

Esta clase debe sobreescribir el método *generatetags*. Este método recibe como parámetro la ruta del archivo real que se está manejando y debe usar la funcione *self.tag* para agregar etiquetas al archivo. La función *self.tag* recibe como parámetros la ruta del archivo, una categoría al que pertenece la etiqueta y la etiqueta en si como tercer parámetro.

Como ejemplo se puede ver `dejumblefs/organizers/date.py` (D.1.12).

C.2.4. Creando un organizador no basado en etiquetas

Para crear un organizador no basado en etiquetas se debe crear una archivo con el nombre del organizador en minúsculas todo, en la carpeta `dejumblefs/organizers/` con extensión `.py`. En este archivo se debe crear una clase Python que tenga por nombre el nombre del organizador terminando en *Organizer* que extienda de *dejumblefs. organizer.Organizer*.

En este caso se deberá sobreescribir *generatepaths* y opcionalmente *increasefilename*.

El método *generatepaths* recibe una ruta al archivo en el sistema de archivos original y debe devolver un iterador, generador, lista o tupla con las rutas convertidas para usar en el espacio de nombres transformado.

El método *increasefilename* se llama en caso de que haya conflicto al usar una ruta de las devueltas por generatepaths. Este método recibe la ruta del archivo en el espacio de nombres transformado y la modifica para evitar el conflicto. Por ejemplo la implementación por defecto de este método transforma “algún archivo.txt” a “algún archivo(1).txt” y en caso de que se vuelva a llamar en “algún archivo(2).txt”.

Como ejemplo se puede ver `dejumblefs/organizers/iso9660.py` (D.1.15).

C.3. Otras consideraciones

Existe un directorio especial en todos los sistemas de archivos creados por DejumbleFS llamado `.dejumblefs`. Dentro de este directorio se encuentran 3 directorios.

root Contiene un proxy transparente a los contenidos de la carpeta que se usó como *root* al momento de montar.

original Contiene un proxy transparente a los contenidos de la carpeta que se usó como *target* al momento de montar.

commands Contiene una lista de comandos que se pueden ejecutar sobre el sistema de archivos, actualmente sólo `umount` que prepara el sistema para ser desmontado. Estos comandos están declarados en la clase *CommandHandler* en el archivo `dejumblefs/fs.py` (D.1.10).

Apéndice D

Código fuente

El siguiente código fuente fue tomado de:

```
URL: https://dejumble.googlecode.com/svn/trunk
Revision: 121
```

D.1. /

D.1.1. dejumblefs/cache.py

```
import errno
import logging
import os

from PyDbLite import Base

from . import util
from .util import Cacheable
from .fs import getserver, CommandHandler

DB_FILES = './.dejumblefs_cache.pydblite'

logger = logging.getLogger('dejumblefs.Cache')


class Cache(Cacheable):
    """
    This is the base class for the caching system
    """

    def __init__(self, filter_):
        self.filter = filter_
        self.files = None
        Cacheable.__init__(self)

    def reset(self):
        self.files = self.files or Base(DB_FILES)
        self.files.create('realpath', mode = 'override')
        self.files.create_index('realpath')
        Cacheable.reset(self)

    def updatecache(self):
        for realpath in self.filter.filelist():
            self.files.insert(realpath)

    def deletefromcache(self, realpath):
        for file in self.files.get_index('realpath')[realpath]:
            self.files.delete(file)

    def addtocache(self, realpath):
        self.files.insert(realpath)

    def filelist(self):
        self.refreshcache()
        for record in self.files:
            yield record['realpath']

    ############################################
    # Original filesystem functions

    def getattr(self, realpath):
        logger.debug('getattr(%s)' % realpath)
        return os.lstat(realpath)

    def readlink(self, realpath):
        logger.debug('readlink(%s)' % realpath)
        return os.readlink(realpath)
```

```

    def unlink(self, realpath):
        logger.debug('unlink(%s)' % realpath)
        os.unlink(realpath)
        self.expirecache()

    def rename(self, realpath, pathdest):
        logger.debug('rename(%s, %s)' % (realpath, pathdest))
        os.rename(realpath, pathdest)
        self.expirecache()

    def chmod(self, realpath, mode):
        logger.debug('chmod(%s, %s)' % (realpath, mode))
        os.chmod(realpath, mode)

    def chown(self, realpath, user, group):
        logger.debug('chown(%s, %s, %s)' % (realpath, user, group))
        os.chown(realpath, user, group)

    def truncate(self, realpath, length):
        logger.debug('truncate(%s, %s)' % (realpath, length))
        ofile = open(realpath, 'a')
        ofile.truncate(length)
        ofile.close()

    def utime(self, realpath, times):
        logger.debug('utime(%s, %s)' % (realpath, times))
        os.utime(realpath, times)

    def access(self, realpath, mode):
        logger.debug('access(%s, %s)' % (realpath, mode))
        if not os.access(realpath, mode):
            return -errno.EACCES

    ############################################
    # File functions embedded in a class

    class DejumbleFile(object):
        """
        This is the base class to manage a File on the caching system.
        """

        def __init__(self, path, flags, *mode):
            logger.debug('DejumbleFile.__init__(%s, %s)' % (path, flags))
            self.keep_cache = False
            self.direct_io = False

            if util.iscommand(path):
                self.file = CommandHandler(path, *mode)
            else:
                self.file = self.getfile(path, flags, *mode)

            if flags & os.O_CREAT:
                getserver().organizer.addtocache(path)

        def getfile(self, path, flags, *mode):
            realpath = getserver().organizer.realpath(path)
            file = open(realpath, util.flags2mode(flags))
            return file

        def read(self, length, offset):
            logger.debug('DejumbleFile.read(%s, %s)' % (length, offset))
```

```
            self.file.seek(offset)
            return self.file.read(length)

        def write(self, data, offset):
            logger.debug('DejumbleFile.write(%s, %s)' % (len(data), offset))
            self.file.seek(offset)
            self.file.write(data)
            return len(data)

        def release(self, flags):
            logger.debug('DejumbleFile.release(%s)' % flags)
            self.file.close()

        def _fflush(self):
            if 'w' in self.file.mode or 'a' in self.file.mode:
                self.file.flush()

        def fsync(self, isfsyncfile):
            logger.debug('DejumbleFile.fsync(%s)' % isfsyncfile)
            if hasattr(self.file, 'fileno'):
                self._fflush()
                if isfsyncfile and hasattr(os, 'fdatasync'):
                    os.fdatasync(self.file.fileno())
                else:
                    os.fsync(self.file.fileno())

        def flush(self):
            logger.debug('DejumbleFile.flush()')
            self._fflush()
            if hasattr(self.file, 'fileno'):
                os.close(os.dup(self.file.fileno()))

        def fgetattr(self):
            logger.debug('DejumbleFile.fgetattr()')
            if hasattr(self.file, 'fileno'):
                return os.fstat(self.file.fileno())

        def ftruncate(self, length):
            logger.debug('DejumbleFile.ftruncate()')
            self.file.truncate(length)
```

D.1.2. dejumblefs/caches/passthrough.py

```
from ..cache import Cache


class PassThroughCache(Cache):
    pass
```

D.1.3. dejumblefs/caches/sandbox.py

```
import os

from ..cache import Cache
from .. import util
from ..fs import getserver


class SandboxCache(Cache):
```

```

    class DejumbleFile(Cache.DejumbleFile):

        def getfdandfile(self, path, flags, *mode):
            # TODO: when changing a file don't open the original but copy
            # and open the copy
            realpath = getserver().organizer.realpath(path)
            fd = os.open(realpath, flags, *mode) #IGNORE:W0142
            file = os.fdopen(fd, util.flags2mode(flags))
            return (fd, file)
```

D.1.4. dejumblefs/filter.py

```
class FileListFilter:
    """
    This class is the base class for file list filters. It gets a file list
    from somewhere and returns it.
    """

    def __init__(self, query, root):
        self.query = query
        self.root = root

    def filelist(self): #IGNORE:R0201
        """
        Returns a file list
        """
        return []
```

D.1.5. dejumblefs/filters/completedirectory.py

```
import os
import os.path

from ..filter import FileListFilter


class CompleteDirectoryFileListFilter(FileListFilter):

    def filelist(self):
        return list(self.generatefilelistrecursive(self.root))

    def generatefilelistrecursive(self, dirname):
        for path in os.listdir(dirname):
            path = os.path.join(dirname, path)
            if os.path.isdir(path) and not os.path.islink(path):
                for realpath in self.generatefilelistrecursive(path):
                    yield realpath
            else:
                yield path
```

D.1.6. dejumblefs/filters/null.py

```
from ..filter import FileListFilter


class NullFileListFilter(FileListFilter):
```

```
    def __init__(self, query='', root=None): #IGNORE:W0613
        FileListFilter.__init__(self, query, '/dev')

    def filelist(self):
        yield '/dev/null'
```

D.1.7. dejumblefs/filters/originaldirectory.py

```
from ..filters.completedirectory import CompleteDirectoryFileListFilter


class OriginalDirectoryFileListFilter(CompleteDirectoryFileListFilter):

    def __init__(self, query=None, root=None): #IGNORE:W0613
        CompleteDirectoryFileListFilter.__init__(self, query, '.')

    def filelist(self):
        return list(self.generatefilelistrecursive('.'))
```

D.1.8. dejumblefs/filters/shell.py

```
import commands
import errno
import os
import re

from ..filter import FileListFilter


class ShellFileListFilter(FileListFilter):

    def filelist(self):
        status, output = commands.getstatusoutput(self.query)

        if status != 0:
            return -errno.ENOENT

        filenames = [re.sub('^%s' % os.getcwd(), '.', o)
                     for o in output.splitlines()]

        return filenames
```

D.1.9. dejumblefs/filters/xesam.py

```
from ..filter import FileListFilter


class XesamFileListFilter(FileListFilter):
    pass
```

D.1.10. dejumblefs/fs.py

```
import errno
import logging
import os
```

```
import platform
from tempfile import NamedTemporaryFile

import fuse


fuse.fuse_python_api = (0, 2)

logger = logging.getLogger('dejumblefs.DejumbleFS')

_SERVER = None


def setserver(server):
    global _SERVER #IGNORE:W0603
    _SERVER = server


def getserver():
    return _SERVER


class DejumbleFS(fuse.Fuse):

    def __init__(self, *a, **kw):
        self.originaldir = None
        self.conf = None
        self.root = None
        self.filter = None
        self.query = None
        self.cache = None
        self.organizer = None
        # HACK: To ignore pylint warnings
        self.parser = None
        self.fuse_args = None
        self.file_class = None
        # end HACK
        fuse.Fuse.__init__(self, *a, **kw) #IGNORE:W0142

    def main(self, *a, **kw):
        logger.info(_('Initializing dejumblefs'))
        self.tempfile = NamedTemporaryFile()
        self.setup_organizer()
        self.file_class = self.organizer.cache.DejumbleFile
        self.originaldir = os.open(self.fuse_args.mountpoint, os.O_RDONLY)
        try:
            profile = False
            if profile:
                import hotshot
                prof = hotshot.Profile("dejumblefs.stats")
                prof.start()
            result = fuse.Fuse.main(self, *a, **kw) #IGNORE:W0142
            if profile:
                prof.stop()
                prof.close()
        except fuse.FuseError:
            result = -errno.ENOENT
            logger.warn(_('Finalizing dejumblefs'))
        return result

    def setoptions(self):
        self.parser.add_option(mountopt="conf",
```

```
                               metavar="CONF",
                               default='~/.dejumblefs/default.xml',
                               help=_("read configuration from CONF file " +
                                      "[default: %default]"))
        self.parser.add_option(mountopt="root",
                               metavar="ROOT",
                               default='.',
                               help=_("root for all file operations " +
                                      "(can be absolute or relative to the " +
                                      "mountpoint) [default: %default]"))
        self.parser.add_option(mountopt="filter",
                               metavar="FILTER",
                               default='OriginalDirectory',
                               help=_("use FILTER to handle QUERY" +
                                      "[default: %default]"))
        self.parser.add_option(mountopt="query",
                               metavar="QUERY",
                               default='',
                               help=_("execute QUERY [default: %default]"))
        self.parser.add_option(mountopt="cache",
                               metavar="CACHE",
                               default='PassThrough',
                               help=_("use CACHE to handle caching " +
                                      "[default: %default]"))
        self.parser.add_option(mountopt="organizer",
                               metavar="ORGANIZER",
                               default='Original',
                               help=_("use ORGANIZER [default: %default]"))

    def setup_organizer(self):
        # HACK: set defaults since fuse is not doing that
        defaults = self.parser.get_default_values()

        self.conf = self.conf or defaults.conf
        self.root = self.root or defaults.root
        self.filter = self.filter or defaults.filter
        self.query = self.query or defaults.query
        self.cache = self.cache or defaults.cache
        self.organizer = self.organizer or defaults.organizer
        # end HACK

        self.root = os.path.expanduser(self.root)

        if self.root.endswith('/'):
            self.root = self.root[:-1]

        filter_ = self._loadclass('filters', 'FileListFilter',
                                  self.filter)(self.query, self.root)
        cache = self._loadclass('caches', 'Cache', self.cache)(filter_)
        self.organizer = self._loadclass('organizers', 'Organizer',
                                         self.organizer)(cache)
        logger.info(_('Done loading modules'))

    def _loadclass(self, moduleprefix, classsuffix, name):
        modulename = 'dejumblefs.%s.%s' % (moduleprefix, name.lower())
        classname = '%s%s' % (name, classsuffix)
        logger.info('Loading %s.%s' % (modulename, classname))
        return getattr(self._import(modulename), classname)

    def _import(self, name):
        mod = __import__(name)
        components = name.split('.')
```

```
        for comp in components[1:]:
            mod = getattr(mod, comp)
        return mod

    def umount(self):
        logger.debug('umount()')
        if platform.system() == 'Darwin':
            # Change directory before umounting
            os.chdir('/tmp')

    ############################################
    # Filesystem functions - general

    def fsinit(self):
        os.fchdir(self.originaldir)

        # HACK: see http://code.google.com/p/dejumble/issues/detail?id=1
        if platform.system() == 'Darwin':
            os.chdir('/tmp')
        # end HACK

        os.close(self.originaldir)
        self.organizer.reset()
        logger.info(_('dejumblefs initialized!'))

    def fsdestroy(self):
        logger.debug('fsdestroy()')
        self.tempfile.close()

    ############################################
    # Filesystem functions - structure

    def getattr(self, path):
        logger.debug('getattr(%s)' % path)
        return self.organizer.getattr(path)

    def readdir(self, path, offset):
        logger.debug('readdir(%s, %s)' % (path, offset))
        return self.organizer.readdir(path, offset)

    def readlink(self, path):
        logger.debug('readlink(%s)' % path)
        return self.organizer.cache.readlink(self.organizer.realpath(path))

    def unlink(self, path):
        logger.debug('unlink(%s)' % path)
        self.organizer.cache.unlink(self.organizer.realpath(path))
        self.organizer.deletefromcache(path)

    def rename(self, path, pathdest):
        logger.debug('rename(%s, %s)' % (path, pathdest))
        self.organizer.cache.rename(self.organizer.realpath(path),
                                    self.organizer.realpath(pathdest))
        self.organizer.deletefromcache(path)
        self.organizer.addtocache(pathdest)

    ############################################
    # Filesystem functions - file attributes

    def chmod(self, path, mode):
        logger.debug('chmod(%s, %s)' % (path, mode))
        self.organizer.cache.chmod(self.organizer.realpath(path), mode)
```

```

    def chown(self, path, user, group):
        logger.debug('chown(%s, %s, %s)' % (path, user, group))
        self.organizer.cache.chown(self.organizer.realpath(path), user, group)

    def truncate(self, path, length):
        logger.debug('truncate(%s, %s)' % (path, length))
        self.organizer.cache.truncate(self.organizer.realpath(path), length)

    def utime(self, path, times):
        logger.debug('utime(%s, %s)' % (path, times))
        self.organizer.cache.utime(self.organizer.realpath(path), times)

    def access(self, path, mode):
        logger.debug('access(%s, %s)' % (path, mode))
        self.organizer.cache.access(self.organizer.realpath(path), mode)


class CommandHandler():

    def __init__(self, path, *mode):
        self.command = getattr(self, os.path.basename(path))
        self.mode = mode

    def seek(self, offset):
        pass

    def read(self, len):
        return None

    def write(self, data):
        logger.debug('CommandHandler.%s.write(%s)' % (self.command, data))
        self.command(data)
        return len(data)

    def flush(self):
        pass

    def truncate(self, len):
        pass

    def open(self):
        pass

    def close(self):
        pass

    ############################################
    # Commands

    COMMANDS = ['umount']

    def umount(self, data):
        getserver().umount()
```

D.1.11. dejumblefs/organizer.py

```
import logging
import os
import os.path
```

```
import re

from PyDbLite import Base
import fuse

from . import util
from .util import Cacheable
from .fs import getserver, CommandHandler


DB_TRANSFORMED = './.dejumblefs_transformed.pydblite'
DB_FILE_TAGS = './.dejumblefs_tags.pydblite'

_INCREASE_REGEX = re.compile('^(.*)\((\d+)\)$')

logger = logging.getLogger('dejumblefs.Organizer')


class Organizer(Cacheable):
    """
    This is the base class for organizers
    """

    def __init__(self, cache, recursive=True):
        Cacheable.__init__(self)
        self.cache = cache
        self.recursive = recursive
        self.transformed = None
        # Do not call reset here, it is called from fs.py when the fs is
        # already started

    def reset(self):
        if not self.transformed:
            self.transformed = Base(DB_TRANSFORMED)
        self.transformed.create('realpath', 'path', 'dirname', mode='override')
        self.transformed.create_index('realpath')
        self.transformed.create_index('path')
        self.transformed.create_index('dirname')
        self.cache.reset()
        Cacheable.reset(self)

    def updatecache(self):
        self.generateallpaths()

    def deletefromcache(self, path):
        realpath = self.realpath(path)
        logger.debug("deletefromcache(%s)" % realpath)
        self.cache.deletefromcache(realpath)
        for item in self.transformed.get_index('realpath')[realpath]:
            self.transformed.delete(item)

    def addtocache(self, path):
        if not self.transformed.get_index('path')[path]:
            realpath = self.realpath(path)
            self.cache.addtocache(realpath)
            self.addfile(realpath)

    ############################################
    # Overwritable functions

    def dirlist(self, path): #IGNORE:W0613
        """
```

```
        Returns a list of (non-existent, generated, virtual) directories for a
        given path. Default implementation.
        """
        return []

    def generatepaths(self, realpath):
        """
        Generates paths for a given real path. A file can have more than one
        transformed path. Default implementation.
        """
        yield util.addtrailingslash(util.removeroot(realpath,
                                                    self.cache.filter.root))

    def generaterealpath(self, path):
        """
        Generates a real path for a inexistent path. Default implementation.
        """
        return os.path.join(self.cache.filter.root, path[1:])

    ############################################
    # General functions

    def generateallpaths(self):
        """
        Generates paths for all the files given by the cache and stores them
        in self.transformed
        """
        for realpath in self.cache.filelist():
            if self.recursive:
                # Add all sub-directories first
                currentpath = self.cache.filter.root

                for pathpart in util.pathparts(util.removeroot(realpath,
                                                  self.cache.filter.root)):
                    currentpath = os.path.join(currentpath, pathpart)
                    self.addfile(currentpath)
            else:
                self.addfile(realpath)

    def addfile(self, realpath):
        """
        Stores a file in self.transformed if not there already and returns the
        paths for that file in the proxy file system
        """
        logger.debug('addfile(%s)' % realpath)
        if not util.ignoretag(util.removeroot(realpath,
                                              self.cache.filter.root)):
            return []

        self.refreshcache()
        transformed = self.transformed.get_index('realpath')[realpath]

        if transformed:
            return (record['path'] for record in transformed)
        else:
            paths = []

            for path in self.paths(realpath):
                while self.transformed.get_index('path')[path]:
                    path = self.increasefilename(path)

                dirname = os.path.dirname(path)
```

```python
                logger.debug('addfile(%s, %s, %s)' % (realpath, path, dirname))
                self.transformed.insert(realpath=realpath, path=path,
                                        dirname=dirname)
                paths.append(path)

            return paths

    def increasefilename(self, filename):
        """
        Returns a new filename in sequence. Called if the current filename
        already exists. This default implementation adds a "(1)" to the end if
        not present or increases that number by one.
        """
        root, ext = os.path.splitext(filename)

        num = 1
        matches = _INCREASE_REGEX.match(root)

        if not matches is None:
            num = int(matches.group(2)) + 1
            filename = matches.group(1)

        return '%s(%i)%s' % (root, num, ext)

    ############################################
    # General functions that read the cache

    def filelist(self, path):
        """
        Returns a list of directories and filenames in a list from cache
        """
        logger.debug('filelist(%s)' % path)
        self.refreshcache()

        for dirname in self.dirlist(path):
            yield dirname

        for record in self.transformed.get_index('dirname')[path]:
            yield os.path.basename(record['path'])

    def paths(self, realpath):
        """
        Generates or returns paths from cache for a given real path
        """
        self.refreshcache()
        paths = self.transformed.get_index('realpath')[realpath]

        if paths:
            return (path['path'] for path in paths)
        else:
            return (path for path in self.generatepaths(realpath))

    def realpath(self, path):
        """
        Returns the real path for a file given the path in the file system.
        """
        logger.debug('realpath(%s)' % path)
        self.refreshcache()
        realpaths = [r['realpath']
                     for r in self.transformed.get_index('path')[path]]

        realpath = None
```

```

        if realpaths:
            realpath = realpaths[0]
        elif path == '/':
            realpath = self.cache.filter.root
        elif path == util.addtrailingslash(util.ORIGINAL_DIR):
            realpath = '.'
        elif util.isspecial(path, 'original', True):
            realpath = os.path.join('.', os.sep.join(util.pathparts(path)[2:]))
        elif util.isspecial(path, 'root', True):
            realpath = os.path.join(self.cache.filter.root,
                                    os.sep.join(util.pathparts(path)[2:]))
        elif util.isspecial(path, 'commands'):
            realpath = '.'
        elif util.iscommand(path):
            realpath = getserver().tempfile.name
        else:
            realpath = self.generaterealpath(path)

        logger.debug('realpath(%s) = %s' % (path, realpath))
        return realpath

    ############################################
    # File system functions

    def getattr(self, path):
        dirname = os.path.dirname(path)
        if util.removeroot(path, os.sep) in self.dirlist(dirname):
            return self.cache.getattr(self.realpath(dirname))
        else:
            return self.cache.getattr(self.realpath(path))

    def readdir(self, path, offset): #IGNORE:W0613
        for filename in util.getbasefilelist():
            yield fuse.Direntry(filename)

        for filename in self._filelist(path):
            yield fuse.Direntry(filename)

    def _filelist(self, path):
        filelist = []
        if path == util.addtrailingslash(util.ORIGINAL_DIR):
            filelist = ['original', 'root', 'commands']
        elif util.isspecial(path, 'root', True):
            filelist = os.listdir(self.realpath(path))
        elif util.isspecial(path, 'original', True):
            filelist = os.listdir(self.realpath(path))
        elif util.isspecial(path, 'commands'):
            filelist = CommandHandler.COMMANDS
        else:
            filelist = self.filelist(path)

        for filename in filelist:
            yield filename


class TagOrganizer(Organizer):

    def __init__(self, cache, category=None):
        self.tags = None
        self.category = category
        Organizer.__init__(self, cache, False)
```

```
    def reset(self):
        if not self.tags:
            self.tags = Base(DB_FILE_TAGS)
        self.tags.create('realpath', 'category', 'tag', mode = 'override')
        self.tags.create_index('realpath')
        self.tags.create_index('category')
        Organizer.reset(self)

    def updatecache(self):
        self._generatetags()
        Organizer.updatecache(self)

    def _deletefromcache(self, path):
        realpath = self.realpath(path)
        logger.debug("_deletefromcache(%s)" % realpath)
        for tag in self.tags.get_index('realpath')[realpath]:
            self.tags.delete(tag)

    def deletefromcache(self, path):
        self._deletefromcache(path)
        Organizer.deletefromcache(self, path)

    def addtocache(self, path):
        self._deletefromcache(path)
        self.generatetags(self.realpath(path))
        Organizer.addtocache(self, path)

    def generatepaths(self, realpath):
        for record in self.tags.get_index('realpath')[realpath]:
            yield os.path.join(os.sep, record['tag'],
                               os.path.basename(realpath))

    def dirlist(self, path):
        if path == '/':
            return self.taglist(self.category)
        else:
            return []

    ############################################
    # Tag functions

    def _generatetags(self):
        for filename in filter(util.ignoretag, #IGNORE:W0141
                               self.cache.filelist()):
            self.generatetags(filename)

    def generatetags(self, filename):
        pass

    def tag(self, realpath, category, tag):
        logger.debug('tag(%s, %s, %s)' % (realpath, category, tag))
        if not tag == None and not tag == '':
            self.tags.insert(realpath, category, tag)

    def filelistbytags(self, category, tags):
        self.refreshcache()
        for record in self.tags.get_index('category')[category]:
            if record['tag'] in tags:
                yield os.path.basename(record['realpath'])

    def taglist(self, category):
```

```
        self.refreshcache()
        return util.unique([record['tag'] for record in
                            self.tags.get_index('category')[category]])
```

D.1.12. dejumblefs/organizers/date.py

```
import os
import time

from ..organizer import TagOrganizer


class DateOrganizer(TagOrganizer):

    def __init__(self, cache):
        TagOrganizer.__init__(self, cache, 'date')

    def generatetags(self, realpath):
        stats = os.stat(realpath)
        lastmod = time.localtime(stats[8])
        today = time.localtime()
        lastweek = time.localtime(time.time() - 7 * 24 * 60 * 60)

        self.tag(realpath, self.category, time.strftime('%Y %B', lastmod))

        if time.strftime('%x', today) == time.strftime('%x', lastmod):
            self.tag(realpath, self.category, _('Today'))

        if time.strftime('%Y%W', today) == time.strftime('%Y%W', lastmod):
            self.tag(realpath, self.category, _('This Week'))

        if time.strftime('%Y%W', lastweek) == time.strftime('%Y%W', lastmod):
            self.tag(realpath, self.category, _('Last Week'))
```

D.1.13. dejumblefs/organizers/documents.py

```
from .. import util
from ..organizer import TagOrganizer


class DocumentsOrganizer(TagOrganizer):

    def __init__(self, cache):
        TagOrganizer.__init__(self, cache, 'filetype')
        self.filetypes = util.readconfig('filetypes')
        for filetype, extensions in self.filetypes.items():
            self.filetypes[filetype] = map(util.extensionregex, #IGNORE:W0141
                                           extensions.split(','))

    def generatetags(self, realpath):
        hastag = False
        for filetype, extensions in self.filetypes.iteritems():
            for extension in extensions:
                if not extension.search(realpath) == None:
                    self.tag(realpath, self.category, _(filetype))
                    hastag = True
        if not hastag:
            self.tag(realpath, self.category, _('Other'))
```

D.1.14. dejumblefs/organizers/flat.py

```
import os.path

from .. import util
from ..organizer import Organizer


class FlatOrganizer(Organizer):

    def __init__(self, cache):
        Organizer.__init__(self, cache, False)

    def generatepaths(self, realpath):
        if not os.path.isdir(realpath):
            yield util.addtrailingslash(os.path.basename(realpath))
```

D.1.15. dejumblefs/organizers/iso9660.py

```
import os
import os.path
import math
import re
import logging

from .. import util
from ..organizer import Organizer

_ISO9660_INCREASE_REGEX = re.compile('^(.*)~(\d+)$')

logger = logging.getLogger('dejumblefs.Organizer')


class ISO9660Organizer(Organizer):

    def generatepaths(self, realpath):
        parts = util.pathparts(util.removeroot(realpath,
                                               self.cache.filter.root))

        if len(parts) <= 1:
            yield util.addtrailingslash(self.convertpath(parts[0]))
        else:
            currentpath = os.sep
            currentrealpath = self.cache.filter.root

            for part in parts[:-1]:
                currentrealpath = os.path.join(currentrealpath, part)
                part = list(self.paths(currentrealpath))[0]
                currentpath = os.path.join(currentpath, part)

            yield os.path.join(currentpath, self.convertpath(parts[-1:][0]))

    def increasefilename(self, filename):
        root, ext = os.path.splitext(filename)

        num = 1
        matches = _ISO9660_INCREASE_REGEX.match(root)

        if not matches is None:
            num = int(matches.group(2)) + 1
```

```
            root = matches.group(1)

        return self.convertpath("%s%s" % (root, ext), num)

    def convertpath(self, filename, num=0):
        root, ext = os.path.splitext(filename)

        # FIXME: exclude all non valid characters
        root = root.replace(' ', '')
        root = root.replace('+', '_')

        size = int(6 - math.log10(len(str(num))))

        if len(root) > size or num > 0:
            if num == 0:
                num = 1
            return "%s~%s%s" % (root.upper()[0:size], num, ext.upper()[0:4])
        else:
            return "%s%s" % (root.upper(), ext.upper()[0:4])
```

D.1.16. dejumblefs/organizers/original.py

```
from ..organizer import Organizer


class OriginalOrganizer(Organizer):
    pass
```

D.1.17. dejumblefs/test/base.py

```
import unittest
import tempfile
import shutil
import os


class BaseFileListFilterTestCase(unittest.TestCase):

    def setUp(self):
        self.original_dir = tempfile.mkdtemp()
        self.mount_dir = tempfile.mkdtemp()
        os.chdir(self.mount_dir)

    def tearDown(self):
        shutil.rmtree(self.original_dir)
        shutil.rmtree(self.mount_dir)
```

D.1.18. dejumblefs/test/filters/completedirectory.py

```
import tempfile

from ..base import BaseFileListFilterTestCase
from ...filters.completedirectory import CompleteDirectoryFileListFilter


class CompleteDirectoryFileListFilterTestCase(BaseFileListFilterTestCase):

```

```
    def testfilelist(self):
        original_file1 = tempfile.mkstemp('', '', self.original_dir)
        original_file2 = tempfile.mkstemp('', '', self.original_dir)
        original_subdir = tempfile.mkdtemp('', '', self.original_dir)
        original_file3 = tempfile.mkstemp('', '', original_subdir)

        filter = CompleteDirectoryFileListFilter('', self.original_dir)
        filelist = list(filter.filelist())

        self.assertEqual(len(filelist), 3)
        self.assertTrue(original_file1[1] in filelist)
        self.assertTrue(original_file2[1] in filelist)
        self.assertTrue(original_file3[1] in filelist)
```

D.1.19. dejumblefs/test/filters/null.py

```
from ..base import BaseFileListFilterTestCase
from ...filters.null import NullFileListFilter


class NullFileListFilterTestCase(BaseFileListFilterTestCase):

    def testfilelist(self):
        filelist = list(NullFileListFilter().filelist())
        self.assertEqual(len(filelist), 1)
        self.assertEqual(filelist[0], '/dev/null')
```

D.1.20. dejumblefs/test/filters/originaldirectory.py

```
import tempfile

from ... import util
from ..base import BaseFileListFilterTestCase
from ...filters.originaldirectory import OriginalDirectoryFileListFilter


class OriginalDirectoryFileListFilterTestCase(BaseFileListFilterTestCase):

    def testfilelist(self):
        original_file1 = tempfile.mkstemp('', '', self.mount_dir)
        original_file2 = tempfile.mkstemp('', '', self.mount_dir)
        original_subdir = tempfile.mkdtemp('', '', self.mount_dir)
        original_file3 = tempfile.mkstemp('', '', original_subdir)

        filter = OriginalDirectoryFileListFilter()
        filelist = list(filter.filelist())

        self.assertEqual(len(filelist), 3)
        self.assertTrue('.%s' % util.removeroot(original_file1[1],
                                                self.mount_dir) in filelist)
        self.assertTrue('.%s' % util.removeroot(original_file2[1],
                                                self.mount_dir) in filelist)
        self.assertTrue('.%s' % util.removeroot(original_file3[1],
                                                self.mount_dir) in filelist)
```

D.1.21. dejumblefs/test/filters/shell.py

```
import os

from ..base import BaseFileListFilterTestCase
from ...filters.shell import ShellFileListFilter


class ShellFileListFilterTestCase(BaseFileListFilterTestCase):

    def testfilelist(self):
        os.chdir('/tmp')
        filelist = list(ShellFileListFilter('echo /dev/null && echo %s/null'
                                            % os.getcwd(), '/').filelist())
        self.assertEqual(len(filelist), 2)
        self.assertTrue(filelist[0], '/dev/null')
        self.assertEqual(filelist[1], './null')
```

D.1.22. dejumblefs/test/organizers/iso9660.py

```
import unittest

from ...organizers.iso9660 import ISO9660Organizer


class ISO9660OrganizerTestCase(unittest.TestCase):

    def setUp(self):
        self.organizer = ISO9660Organizer(None)

    def testincreasefilename(self):
        self.assertEquals(self.organizer.increasefilename("A.TXT"),
                          "A~1.TXT")
        self.assertEquals(self.organizer.increasefilename("A~1.TXT"),
                          "A~2.TXT")
        self.assertEquals(self.organizer.increasefilename("123456~9.TXT"),
                          "12345~10.TXT")

    def test_path(self):
        self.assertEquals(self.organizer.convertpath("a.txt"), "A.TXT")
        self.assertEquals(self.organizer.convertpath("1+2 345.txt"),
                          "1_2345.TXT")
        self.assertEquals(self.organizer.convertpath("1234567890.text"),
                          "123456~1.TEX")
```

D.1.23. dejumblefs/test/util.py

```
import os
import unittest

from .. import util


class UtilTestCase(unittest.TestCase):

    def testpathparts(self):
        self.assertEquals(len(util.pathparts('')), 0)
        self.assertEquals(len(util.pathparts('/')), 1)
        parts = util.pathparts('/a/b/c')
```

```
        self.assertEquals(len(parts), 3)
        self.assertEquals(parts[0], 'a')
        self.assertEquals(parts[1], 'b')
        self.assertEquals(parts[2], 'c')

    def testflags2mode(self):
        self.assertEqual(util.flags2mode(os.O_RDONLY), 'r')
        self.assertEqual(util.flags2mode(os.O_WRONLY), 'w')
        self.assertEqual(util.flags2mode(os.O_RDWR), 'w+')
        self.assertEqual(util.flags2mode(os.O_RDONLY), 'r')
        self.assertEqual(util.flags2mode(os.O_WRONLY | os.O_APPEND), 'a')
        self.assertEqual(util.flags2mode(os.O_RDWR | os.O_APPEND), 'a+')

    def testaddtrailingslash(self):
        self.assertEquals(util.addtrailingslash('a/b/c'), '/a/b/c')
        self.assertEquals(util.addtrailingslash('/a/b/c'), '/a/b/c')
        self.assertEquals(util.addtrailingslash('/'), '/')
        self.assertEquals(util.addtrailingslash(''), '/')

    def testignoretag(self):
        self.assertTrue(util.ignoretag('/abc'))
        self.assertFalse(util.ignoretag('/..'))
        self.assertFalse(util.ignoretag('/.'))
        self.assertFalse(util.ignoretag('/.dejumblefs'))

    def testgetbasefilelist(self):
        self.assertTrue('..' in util.getbasefilelist())
        self.assertTrue('.' in util.getbasefilelist())
        self.assertEquals(len(util.getbasefilelist()), 2)

    def testunique(self):
        list1 = [5, 2, 3, 4]
        list2 = [5, 2, 3, 4, 5, 3]
        self.assertEquals(str(sorted(util.unique(list1))),
                          str(sorted(list1)))
        self.assertEquals(str(sorted(util.unique(list2))),
                          str(sorted(list1)))
        self.assertNotEquals(str(sorted(util.unique(list2))),
                             str(sorted(list2)))
```

D.1.24. dejumblefs/ui/dejumble.py

```
#!/usr/bin/env python

import sys
import logging
import logging.config
import pkg_resources
import gettext
import errno

import fuse
from fuse import Fuse

import dejumblefs.fs

gettext.install('dejumblefs')

try:
    import psyco
```

```
    psyco.full()
except ImportError:
    pass


def main():
    usage = """
dejumble: presents the content of a directory in an organized structure.

""" + Fuse.fusage

    dolog = True

    server = dejumblefs.fs.DejumbleFS(version="%%prog %s" % fuse.__version__,
                                      usage=usage, dash_s_do='setsingle')
    server.setoptions()
    server.parse(values=server, errex=1)

    if not server.fuse_args.mountpoint:
        print >> sys.stderr, (_("No mountpoint defined"))
        sys.exit(-errno.ENOENT)

    if dolog:
        filename = pkg_resources.resource_filename('dejumblefs',
                                                   'conf/logging.conf')
        logging.config.fileConfig(filename)
        # redirect stdout to a disk file
        saveout = sys.stdout
        saveerr = sys.stderr
        outfile = open('/tmp/log.txt', 'a+')
        sys.stdout = outfile
        sys.stderr = outfile
    else:
        logging.disable(logging.CRITICAL)

    dejumblefs.fs.setserver(server)
    server.main()

    if dolog:
        # restore stdout
        outfile.flush()
        outfile.close()
        sys.stdout = saveout
        sys.stderr = saveerr


if __name__ == '__main__':
    main()
```

D.1.25. dejumblefs/ui/dejumblegui.py

```
#!/usr/bin/env python

from __future__ import with_statement

import gettext
import commands
import sys
import os.path
import pickle
```

```
import pkg_resources

import wx

from dejumblefs import util

gettext.install('dejumblefs')

_TB_NEW = 1
_TB_OPEN = 2
_TB_SAVE = 3
_TB_MOUNT = 4
_TB_UMOUNT = 5
_TITLE = _('DejumbleFS Mounter')
_EXTENSION = 'dfo'
_DEJUMBLE_FILES = _('DejumbleFS options') + '(*.%s)|*.%s' % (_EXTENSION,
                                                              _EXTENSION)


class DejumbleFSUI(wx.App):

    def OnInit(self):
        self.main = MainWindow()
        self.main.Show()
        self.TopWindow = self.main

        return True


class MainWindow(wx.Frame):

    def __init__(self):
        wx.Frame.__init__(self, None, title=_TITLE,
                          style=wx.CAPTION|wx.CLOSE_BOX)

        # FIXME: set icon
        #iconpath = pkg_resources.resource_filename('dejumblefs.ui',
        #                                           'images/icon.png')
        #self.Icon = wx.IconFromLocation(wx.IconLocation(iconpath))
        self.panel = wx.Panel(self)

        externalborder = 10
        internalborder = 3

        self.vbox = vbox = wx.BoxSizer(wx.VERTICAL)

        ##################################
        # Mountpoint Options
        sizer = wx.FlexGridSizer(1, 2, hgap = 5)

        label = wx.StaticText(self.panel, label=_('Mount point:'),
                              size=(100, -1), style=wx.ALIGN_RIGHT)
        self.mountpoint = wx.DirPickerCtrl(self.panel, size=(300, -1))
        self.mountpoint.Bind(wx.EVT_DIRPICKER_CHANGED, self._setenabledall)
        sizer.Add(label, flag=wx.ALIGN_CENTER_VERTICAL)
        sizer.Add(self.mountpoint, flag=wx.ALL, border=internalborder)

        self.nonempty = wx.CheckBox(self.panel, label='nonempty')
        sizer.AddSpacer(0)
        sizer.Add(self.nonempty, flag=wx.ALL, border=internalborder)

        self.noappledouble = wx.CheckBox(self.panel, label='noappledouble')
```

```
        sizer.AddSpacer(0)
        sizer.Add(self.noappledouble, flag=wx.ALL, border=internalborder)

        vbox.Add(sizer, flag=wx.ALL, border=externalborder)
        vbox.Add(wx.StaticLine(self.panel), 0, wx.ALL|wx.EXPAND)

        ##################################
        # Filter Options
        sizer = wx.FlexGridSizer(1, 2, hgap = 5)

        choices = ['CompleteDirectory', 'OriginalDirectory', 'Null', 'Shell']
        label = wx.StaticText(self.panel, label=_('Filter:'),
                              size=(100, -1), style=wx.ALIGN_RIGHT)
        self.filter = wx.ComboBox(self.panel, choices=choices,
                                  style=wx.CHOICEDLG_STYLE,
                                  value=choices[0])
        sizer.Add(label, flag=wx.ALIGN_CENTER_VERTICAL)
        sizer.Add(self.filter, flag=wx.ALL, border=internalborder)

        label = wx.StaticText(self.panel, label=_('Root:'),
                              size=(100, -1), style=wx.ALIGN_RIGHT)
        self.root = wx.DirPickerCtrl(self.panel, size=(300, -1))
        self.root.TextCtrlGrowable = True
        self.root.PickerCtrlProportion = 0.1
        self.root.TextCtrlProportion = 0.1
        sizer.Add(label, flag=wx.ALIGN_CENTER_VERTICAL)
        sizer.Add(self.root)

        label = wx.StaticText(self.panel, label=_('Query:'),
                              size=(100, -1), style=wx.ALIGN_RIGHT)
        self.query = wx.TextCtrl(self.panel, size=(300, -1))
        sizer.Add(label, flag=wx.ALIGN_CENTER_VERTICAL)
        sizer.Add(self.query, flag=wx.ALL, border=5)

        vbox.Add(sizer, flag=wx.ALL, border=externalborder)
        vbox.Add(wx.StaticLine(self.panel), 0, wx.ALL|wx.EXPAND)

        ##################################
        # Other Options
        sizer = wx.FlexGridSizer(1, 2, hgap = 5)

        choices = ['PassThrough', 'Sandbox']
        label = wx.StaticText(self.panel, label=_('Cache:'),
                              size=(100, -1), style=wx.ALIGN_RIGHT)
        self.cache = wx.ComboBox(self.panel, choices=choices,
                                 style=wx.CHOICEDLG_STYLE,
                                 value=choices[0])
        sizer.Add(label, flag=wx.ALIGN_CENTER_VERTICAL)
        sizer.Add(self.cache, flag=wx.ALL, border=internalborder)

        choices = ['Original', 'Flat', 'ISO9660', 'Documents', 'Date']
        label = wx.StaticText(self.panel, label=_('Organizer:'),
                              size=(100, -1), style=wx.ALIGN_RIGHT)
        self.organizer = wx.ComboBox(self.panel, choices=choices,
                                     style=wx.CHOICEDLG_STYLE,
                                     value=choices[0])
        sizer.Add(label, flag=wx.ALIGN_CENTER_VERTICAL)
        sizer.Add(self.organizer, flag=wx.ALL, border=internalborder)

        vbox.Add(sizer, flag=wx.ALL, border=externalborder)

        ##################################
```

```
        # Layout and other

        self._createtoolbar()

        hbox = wx.BoxSizer(wx.HORIZONTAL)
        hbox.Add(vbox, flag=wx.ALL, border=10)
        self.panel.Sizer = hbox
        hbox.Fit(self)
        self._setenabledall()

        self.new()
        self.Center()

    def _createtoolbar(self):
        self.ToolBar = wx.ToolBar(self,
                                  style=wx.TB_TEXT|wx.TB_HORIZONTAL|wx.TB_TOP)

        img = wx.ArtProvider.GetBitmap(wx.ART_NEW)
        self.ToolBar.AddLabelTool(1, _('New'), img)
        self.ToolBar.AddSeparator()
        img = wx.ArtProvider.GetBitmap(wx.ART_FILE_OPEN)
        self.ToolBar.AddLabelTool(2, _('Open'), img)
        img = wx.ArtProvider.GetBitmap(wx.ART_FILE_SAVE)
        self.ToolBar.AddLabelTool(3, _('Save'), img)
        self.ToolBar.AddSeparator()
        img = wx.ArtProvider.GetBitmap(wx.ART_NEW_DIR)
        self.ToolBar.AddLabelTool(4, _('Mount'), img)
        img = wx.ArtProvider.GetBitmap(wx.ART_DELETE)
        self.ToolBar.AddLabelTool(5, _('Umount'), img)
        self.ToolBar.Realize()

        self.Bind(wx.EVT_TOOL, self.new, id=_TB_NEW)
        self.Bind(wx.EVT_TOOL, self.open, id=_TB_OPEN)
        self.Bind(wx.EVT_TOOL, self.save, id=_TB_SAVE)
        self.Bind(wx.EVT_TOOL, self.mount, id=_TB_MOUNT)
        self.Bind(wx.EVT_TOOL, self.umount, id=_TB_UMOUNT)

    def new(self, event=None, mountpoint='', nonempty=False,
            noappledouble=False, filter=None, root='', query='',
            cache=None, organizer=None, filename=None):
        self.mountpoint.Path = mountpoint
        self.nonempty.Value = nonempty
        self.noappledouble.Value = noappledouble
        self.root.Path = root
        self.query.Value = query

        if filter:
            self.filter.Value = filter
        else:
            self.filter.Select(0)

        if cache:
            self.cache.Value = cache
        else:
            self.cache.Select(0)

        if organizer:
            self.organizer.Value = organizer
        else:
            self.organizer.Select(0)

        self.filename = filename
```

```
        self._settitle()

    def open(self, event):
        dialog = wx.FileDialog(self, wildcard=_DEJUMBLE_FILES)
        dialog.ShowModal()
        filename = dialog.Path
        if filename:
            with open(filename, 'rb') as file:
                result = pickle.load(file)
            self.new(filename=filename, **result)
            self._setenabledall()

    def save(self, event):
        if not self.filename:
            dialog = wx.FileDialog(self, wildcard=_DEJUMBLE_FILES,
                                   style=wx.FD_SAVE)
            dialog.ShowModal()
            filename = dialog.Path
            if not filename:
                return
            if filename.endswith('.%s.' % _EXTENSION):
                filename = filename[:-1]
            if filename.endswith('.'):
                filename = filename + _EXTENSION
            elif not filename.endswith('.%s' % _EXTENSION):
                filename = filename + '.%s' % _EXTENSION
            self.filename = filename
            self._settitle()

        with open(self.filename, 'wb') as file:
            values = {'mountpoint': self.mountpoint.Path,
                      'nonempty': self.nonempty.Value,
                      'noappledouble': self.noappledouble.Value,
                      'filter': self.filter.Value,
                      'root': self.root.Path,
                      'query': self.query.Value,
                      'organizer': self.organizer.Value}
            pickle.dump(values, file)

    def _settitle(self):
        if self.filename:
            self.Title = '%s - %s' % (_TITLE,
                                      self.filename.split(os.path.sep)[-1])
        else:
            self.Title = '%s - %s' % (_TITLE, 'Untitled')

    def mount(self, event):
        flags = []

        if self.nonempty.Value:
            flags.append(",nonempty")

        if self.noappledouble.Value:
            flags.append(",noappledouble")

        command = 'dejumble "%s" -o root="%s",query="%s",' \
                  'filter="%s",cache="%s",organizer="%s"%s' % \
                  (self.mountpoint.Path, self.root.Path, self.query.Value,
                   self.filter.Value, self.cache.Value, self.organizer.Value,
                   ''.join(flags))

        status, output = commands.getstatusoutput(command)
```

```

        if output:
            wx.MessageDialog(self, "Error mounting: %s" % output, 'Error',
                             wx.OK | wx.ICON_ERROR).ShowModal()

        self._setenabledall()

    def umount(self, event):
        command = 'umountdejumble "%s"' % self.mountpoint.Path

        status, output = commands.getstatusoutput(command)

        if output:
            wx.MessageDialog(None, 'Error umounting: %s' % output, 'Error',
                             wx.OK | wx.ICON_ERROR).ShowModal()

        self._setenabledall()

    def _setenabledall(self, event=None):
        enable = not os.path.isdir(os.path.join(self.mountpoint.Path,
                                                util.ORIGINAL_DIR))

        for child in self.Children:
            child.Enabled = enable

        self.ToolBar.EnableTool(_TB_MOUNT, enable)
        self.ToolBar.EnableTool(_TB_UMOUNT, not enable)


def main():
    application = DejumbleFSUI(0)
    application.MainLoop()


if __name__ == '__main__':
    main()
```

D.1.26. dejumblefs/ui/images/createicon.py

```
from __future__ import division

import random

_COLORS = ['#0099cc', '#006699']

def main():
    masks = []
    squares = []
    size = 128
    squarewidth = 10
    for x in range(0, size + 1, squarewidth):
        for y in range(0, size + 1, squarewidth):
            color = _COLORS[((x + y) // squarewidth) % 2]
            id = 990000 + (100 * x) + y
            d = (4 * random.randint(0, size - x) // size) ** 2
            dx = random.randint(-d, d)
            dy = random.randint(-d, d)

            masks.append('    <mask id="mask%i">'
                         '<circle cx="%i" cy="%i" r="%i" fill="#ffffff"/>'
```

```
                         '</mask>' % (id, size // 2 + dx, size // 2 + dy, size // (2 / 0.8)))

            squares.append('    <rect style="fill:%s" id="rect%i" width="%i" '
                           'height="%i" x="%i" y="%i" mask="url(#mask%s)" />'
                           % (color, id, squarewidth, squarewidth,
                              x + dx, y - 1 + dy, id))

    print('<?xml version="1.0" encoding="UTF-8" standalone="no"?>')
    print('<svg width="%ipx" height="%ipx">' % (size, size))
    print('  <defs>')
    print('\n'.join(masks))
    print('  </defs>')
    print('  <g>')
    print('\n'.join(squares))
    print('  </g>')
    print('</svg>')

if __name__ == '__main__':
    main()
```

D.1.27. dejumblefs/ui/umountdejumble.py

```
#!/usr/bin/env python

import commands
import sys
import errno
import os.path
import gettext
import platform

from .. import util

gettext.install('dejumblefs')


def main():
    commandname = os.path.basename(sys.argv[0])
    if len(sys.argv) == 1:
        print "usage: %s [mountpoint]" % commandname
        sys.exit(1)

    path = sys.argv[1]
    command_path = os.path.join(path, util.ORIGINAL_DIR, 'commands', 'umount')

    if not os.path.isdir(os.path.join(path, util.ORIGINAL_DIR)):
        print >> sys.stderr, _('%s: %s: not a dejumble filesystem') % \
              (commandname, path)
        sys.exit(-errno.ENOENT)

    status, output = commands.getstatusoutput('echo 1 > "%s"' % command_path)

    if status != 0:
        print >> sys.stderr, _('%s: %s') % (commandname, output)
        sys.exit(status)

    if platform.system() == 'Darwin':
        status, output = commands.getstatusoutput('umount "%s"' % path)
    else:
        status, output = commands.getstatusoutput('fusermount -u "%s"' % path)
```

```

    if status != 0:
        print >> sys.stderr, _('%s: %s') % (commandname, output)
        sys.exit(status)


if __name__ == '__main__':
    main()
```

D.1.28. dejumblefs/util.py

```
import os
import re
import time
import logging

from pkg_resources import resource_filename #IGNORE:E0611

ORIGINAL_DIR = '.dejumblefs'

logger = logging.getLogger('dejumblefs.DejumbleFS')


def pathparts(path):
    return path.split('/')[1:]


def flags2mode(flags):
    filemode = {os.O_RDONLY: 'r', os.O_WRONLY: 'w', os.O_RDWR: 'w+'}
    mode = filemode[flags & (os.O_RDONLY | os.O_WRONLY | os.O_RDWR)]
    if flags & os.O_APPEND:
        mode = mode.replace('w', 'a', 1)
    return mode


def addtrailingslash(path):
    if path.startswith(os.sep):
        return path
    else:
        return '%s%s' % (os.sep, path)


def removeroot(realpath, root):
    if realpath.startswith(root):
        return realpath.replace(root, '', 1)
    else:
        raise RuntimeError


def ignoretag(filename):
    return (not filename == '/..' and not filename == '/.'
            and not filename.startswith('/.dejumblefs'))


def extensionregex(extension):
    return re.compile('%s$' % extension)


def getbasefilelist():
    return ['..', '.']
```

```


def unique(string):
    return set(string)


def iscommand(path):
    return pathparts(path)[0:2] == [ORIGINAL_DIR, 'commands']


def isspecial(path, dir, includesubdirs=False):
    if includesubdirs:
        return pathparts(path)[0:2] == [ORIGINAL_DIR, dir]
    else:
        return path == addtrailingslash(os.path.join(ORIGINAL_DIR, dir))

#############################################
# Cacheable class


class Cacheable:

    def __init__(self):
        self.expiretime = time.time()

    def reset(self):
        self.expirecache()
        self.refreshcache()

    def expirecache(self):
        self.expiretime = time.time()

    def refreshcache(self):
        if self.expiretime < time.time():
            self.expiretime = time.time() + 60
            self.updatecache()

    def updatecache(self):
        pass

    def deletefromcache(self, string):
        pass

    def addtocache(self, string):
        pass

#############################################
# Configuration functions

_CONFIGURATION = {}


def readconfig(name):
    if name not in _CONFIGURATION:
        defaultfilename = resource_filename('dejumblefs',
                                            'conf/%s-default.conf' % name)
        userfilename = os.path.expanduser('~/.dejumblefs/%s.conf' % name)
        currentdirfilename = './.dejumblefs/%s.conf' % name
        config = {}
        readconfigfile(config, defaultfilename)
        readconfigfile(config, userfilename)
        readconfigfile(config, currentdirfilename)
```

```
            _CONFIGURATION[name] = config

    return _CONFIGURATION[name]


def readconfigfile(config, path):
    if os.path.isfile(path):
        ofile = open(path, 'r')
        for line in ofile.readlines():
            name, value = line.split('=', 1)
            config[name.strip()] = value.strip()

    return config
```

D.2. /docs/thesis/Chapter4/Chapter4Figs

D.2.1. general.gnu.inc

```
set terminal pdf
set data style histograms
set style fill solid 0.25 border -1
set key autotitle columnheader
set encoding iso_8859_1
```

D.2.2. pruebaA.gnu

```
load "general.gnu.inc"
set output FILE.'.eps'
set ylabel "Microsegundos"
set xlabel "Numero de archivos"

plot FILE.'.dat' using 2, '' using 3:xticlabels(1)
```

D.2.3. pruebaB.gnu

```
load "general.gnu.inc"
set output FILE.'.eps'
set ylabel "Microsegundos"
set xlabel "Tamaño del archivo"

plot FILE.'.dat' using 2, '' using 3:xticlabels(1)
```

Referencias

[ea99] Neil Brown et al. The linux virtual file-system layer. http://www.cse.unsw.edu.au/~neilb/oss/linux-commentary/vfs.html, December 1999. 7

[fus08] FUSE: File system in user space. http://fuse.sourceforge.net/, March 2008. 3

[gpl08] The GNU general public license - GNU project - free software foundation (FSF). http://www.gnu.org/licenses/gpl.html, March 2008. 3

[Hen08] Val Henson. A brief history of UNIX file systems. http://www.valhenson.org/fs_slides.pdf, March 2008. 1, 5

[lin08] Lxr linux. http://lxr.linux.no/linux+v2.6.27.6, March 2008. 6

[pos08] The open group – single UNIX specification version 3. http://www.unix.org/single_unix_specification/, March 2008. 4

[psy08] Psyco - home page. http://psyco.sourceforge.net/, March 2008. 3

[pyt08] Python programming language – official website. http://www.python.org/, March 2008. 3

[vfs08] The linux virtual file-system layer: Inodes and operations. http://www.cse.unsw.edu.au/~neilb/oss/linux-commentary/vfs-7.html, March 2008. 6

[xes08] Frontpage - XESAM wiki. http://xesam.org/, March 2008. 4